THE ANALYSIS AND COMPARISON OF SYSTEM DEVELOPMENT METHODOLOGIES IN SOFTWARE ENGINEERING

Thesis submitted in accordance with the requirements of the University of Liverpool, United Kingdom, for the degree of Doctor in Philosophy

By

Maqbool Uddin Shaikh

December 1986.

DEDICATION

This Thesis is dedicated to my late parents Siraj Uddin, Khurshed Begum, my wife Khalida and my childern Saima and Uzair for their love, support and patience.

ACKNOWLEDGEMENTS

I wish I could know the special words to thank Professor Michael Anthony Hennell who always found confidence in me, since I was doing M.Sc. with him, and giving me a chance to work on his projects as a Research Assistant at the University of Liverpool, UK. I am grateful for his unwavering patience, guidence and supervision throughout my student life at the university of Liverpool.

I am also thankful to the European Research Office, Science and Engineering Research Council, UK., and Liverpool Data Research Associates, for financing this research.

My special thanks to Mrs. Margret Hennell, the Managing Director of Liverpool Data Research Associates, for reading this thesis. It was her kindness to offer valueable comments and help me with patience to make the language and presentation of the thesis better and better.

I would like to express my gratitude to my late parents for their influence, guidence and support which they offered in their life time. They gave me training in how to adopt the cognitive style in acquiring knowledge. I wish they were alive to see their desire come true.

Finlly I want to thank to my wife Khalida who always encouraged and gave moral support to me all long my path to the Ph.D.

<u>*CONTENTS*</u>

Page

CHAPTER ONE

1.1 Introduction

In recent years many Systems Analysis & Design Methodologies have been developed as a result of market Requirements in different problem areas of software engineering. These Methodologies differ widely in features such as approach, concepts, coverage, complexity, terminology, tools etc.. They are mainly problem and environment oriented and therefore are very good for certain types of problems, but unsuitable for other classes. In spite of the commercial claims most of these Methodologies do not cover the whole span of System development and completely ignore certain Phases or activities of System development. For example MASCOT which is called a Methodology by its authors totally ignores any Analysis activities.

It is difficult for the Analyst/Designer and the User/Management to decide which Methodology should be adopted to solve their problems. The reason for this is that:.

1. These Methodologies were initially developed for certain specific purposes, usage, environments etc. and later, due to commercial reasons, were modified to be used in other areas.
2. These Methodologies use different terminology for the same things, and use

similar names for different things. Such discrepencies in terminology cause confusion.

3. More often these Methodologies use certain terms such as function, process, procedure, action, event, entity etc. without explicit definition.

To compare & evaluate these Methodologies their features, facilities, etc. is not an easy job. The following are a few possible approaches; some of them were applied previously and others are applied in this study:.

1. Use the Methodologies to solve a model problem related to a particular environment and compare how successfuly the features and facilities of these Methodologies can be applied to solve that problem.
2. Develop a list of model questions and apply it on selected Methodologies to identify and study their characteristics and features. Use the selected important characteristics & features to compare & evaluate the individual Methodologies.
3. Carry out a feature Analysis of the Methodologies by using a set of questions uniformly on each selected Methodology with out doing a comparative Analysis, and express these features in uniform

terminologies.

4. Compare the Methodologies against an ideal one. A Methodology which is commonly used and applicable to all different types of the problems.

5. Compare the Methodologies quantitatively i.e. in terms of metrics, indices, statistical measures, etc..

6. Develop a set of questions and apply them to the individual Methodologies to determine their objectives, scope, characteristics, features, etc.. Use the Phases of the Life Cycle as standard activities to examine, determine and evaluate the extent to which each Methodology covers the Phases of the Life Cycle i.e. cover the span of System development tasks. Compare the characteristics, features, etc.. of the Methodologies with each other as far as possible. This comparision is of a relative nature. This approach is a modification of the second approach and is one of the two approaches which are adopted in our study.

7. Study how the terminologies are defined and used by the selected Methodologies. Further compare their definitions with the IEEE standard definitions, the glossary

developed here and the definitions given by the other Methodologies for such terms as function, procedure, process, action, state, entity etc.. This is the second approach adopted in this thesis.

The first approach in this list was adopted in the early eighties by a large research group whose members were from various selected organizations [78]. One of the purposes of this group was to study the existing Methodologies and identify the most suitable and beneficial Methodologies for the Ada environment. Two model examples were created by the group for partial development using the selected Methodologies. The first example was an aircraft monitoring system. CORE, MASCOT, JSD, SARA & CSS were applied to develop a number of subsystems. The second problem was to develop a KAPSE database by using VDM, (software development ideas given by Jones) [30] and also abstract data types. The output systems for the above problems could not be implemented due to lack of access to suitable compilers. So it is difficult to assess the success of this study. The span of the Life Cycle model used by the group has limited scope and coverage. Many elements covered in the sample and referred to as a Methodology are not a Methodology according to the defination in the glossary. In fact these are a subset of a Methodology because they do not cover the whole span of the System development tasks. For example, the authors of JSD call it a Method, not a Methodology.

A generalised form of the above approach was adopted by IFIP WG. 8.1 during 1982 [63]. The working group developed a model problem about a hypothetical conference organisation to be solved by various existing Methodologies. The aim of the organization was to find the strong and weak points of each Methodology and compare their solutions of the model problem. The WG. decided to send an open invitation along with a brief Requirement Specifications for having a computerised System of the model problem; to the various Methodology authors. The authors were asked to submit a paper explaining how they can apply and solve the model problem which was also called a standard test case. Authors of twenty-five Methodologies submitted their papers in the conference. A review committee compared these submissions, without attempting to be too analytic, and examined the facilities of the Methodologies which they applied to solve the model problem. Later the committee selected seven Methodologies which best cover the existing spectrum of Methodologies. The spectrum was defined informally as the process/functions to be performed and the Analysis of data to be used in the System. Then thirteen out of twenty-five Methodologies were selected for inclusion in the proceedings of the first conference. The proceedings show the similarities and differences among the Methodologies which came into prominence after applying the test case. This approach had many drawbacks. For example practically it was difficult to develop a problem which could equally be used by these Methodologies to show their facilities and capabilities to solve the particular problem. The term Methodology itself was not defined so the authors of Methods were also able to take part in the study. There was a great risk that the model problem no matter how carefully it was designed may be biased towards or against a particular Methodology. The reason for this is that all these Methodologies are more or less problem oriented and are not suitable to all types of problems. Also the model problem was found to be too well defined and did not include the development of the Requirements Specification. This limited the coverage of the System development tasks, ignoring important Phases. Hence it becomes inadvisable to use a model problem to compare the features of the Methodologies. Such an approach will not be used in this study.

The second approach was adopted by several groups separately. First it was also adopted during 1981 by the research group mentioned above whose main purpose was to study an Ada based System development Methodology [78]. A framework was developed to identify a set of 13 characteristics against which 21 Methodologies or their variants were examined. One of these characteristics was to study Ada compatibility ie. how Ada might be used by these Methodologies. Further this group also studied how successfully these Methodologies can solve problems related to Ada environments. Later during 1982, Wasserman and his group conducted a similar survey of 24 'software development methodologies' [82]. One of the purposes of this study was to determine which 'methodologies', if there were any, could be integrated with the Ada language, and the related Ada programming support environment. The framework of this study was objective and was provided by the Ada joint program office. A questionaire was used to obtain basic information on the existing 'software development methodologies', particularly as they may be used in conjunction with programs to be written in Ada [83]. In this sample they also included some elements which according to their own definition of 'methodology' [p37-92] are Methods such as JSD, SD. As these two studies were purposive and biased towards Ada, their results may not be fair and general.

A similar approach to study the Methodologies was also adopted by IFIP. In their 2nd. conference of WG. 8.1, during July 1983 [64]. The aim of the second conference was "to carry out a feature analysis of some of the methodologies from the first stage, plus any other methodologies that contributors might wish to include" [79]. In this conference the solutions of the model problem of the 1st. conference, given by different Methodologies were compared and analysed by various participants. The comparision and Analysis of these Methodologies was done against a number of features and on the basis of the information about their goals, applicibility, origin, experience, representation means etc.. This information was specified in their submission, for the first conference. In this second conference different participants tried to compare and analyse the features of the Methodologies without having a common framework or a list of features. It is therefore, difficult to relate their work with each other and find common points of interest on which all the researchers can agree. The definition of Methodology, what constitute a Methodology, was not defined by the organisors. The following is a brief description of the studies which were done by several participants separately.

Brandt [80] selected thirteen Methodologies to do a feature Analysis. These Methodologies were selected from the twenty-five Methodologies which appeared in the proceedings of the 1st. IFIP WG 8.1 working on the Comparative Review of Information System Design Methodologies Conference [63]. His study was based on the assumptions, goals and main application areas stated in the proceedings. Efforts were made to study the stated objectives of these Methodologies and tried to attain an overview of the eight selected major features such as origin & experience, development Phases, tools, documentation, etc.. This study was based on the assumptions, goals and main application areas stated in the 1st conference, so it had an impact of all the previous drawbacks. Brandt comments that "an exhaustive Analysis has not been attempted" [p-9,80]. This study did not cover each Methodology in depth. For example how the Methodologies achieve their objectives were not considered. Selected features to do the Analysis were not sufficient and did not cover the complete span of System development tasks. The selected Methodologies were mainly concerned with or limited to the theoretical university academic environment, without practical experience. some of them were found to be still in the research stage. No Methodology from their sample is included in our study.

The survey conducted by Wasserman and his group also studied the selected 'methodologies' according to the second approach [81]. In this survey efforts were also made to study the 'methodologies' against certain selected parameters concerned with the software development, and determine to what extent these parameters are covered by the 'methodologies'. The scope of the survey was therefore, biased towards compatibility with the Ada language. Wasserman et al have themselves commented on the study as "the objective questions, while designed to be easier to report, did not provide as much useful information as we had hoped. We often felt that our terms were misinterpreted or that the developer of the methodology did not relate accurately to a question". ...." ....the loosely structured answers made the job of reporting the results more difficult." [p-81, 83]. Further the information about the 'methodologies' were provided by the authors of those 'methodologies', so due to commercial reasons the honesty, accuracy and reliability of their answers cannot be guaranteed. Of course no one will like to answer negatively exposing the weak points of his 'methodology' when they are in the market. The group has admitted that "we made little effort to evaluate the claims of the respondents" [p-41, 81]. The entries in most of the tables are according to the way the terminologies have been defined by the 'methodologies'. This is another weak point of their study because these terminologies and their use differ from 'methodology to methodology'. Porcella et al., have admitted that "another problem was confusion about understanding whether what we meant by 'requirement' and 'specification' in the questionaire was what the respondent meant" [p-84, 83]. In this study only three out of the 24 'methodologies' referred to by Wasserman are covered.

Olive [84] proposed a frame work to do a comparative Analysis of two of the aspects of the System Design Methodologies i.e. level of abstraction and type of Methodologies. The term abstraction is closely related to the hierarichal decomposition of a complex System. Five different levels of abstraction i.e. external, conceptual, logical, architectural and physical, were used in this study. The information Systems were classified as data base Systems, decision support Systems, etc.. The main purpose of this framework was to find common points i.e. similarities among the Methodologies rather than differences. The evaluation of the Methodologies was out of the scope of this study. This approach was therefore, incomplete. Only eleven Methodologies were covered, all of them were presented in the proceedings of the 1st. conference [63]. None of these

Methodologies is covered in our study.

A similar study based on feature Analysis was also made by Iivari and his group [85]. A frame work based on a sociocybernetic interpretation of System Design, human action and System Design Methodologies was prepared for this purpose. Features like scope & content, support & usability, assumptions, etc.. were studied. The feature Analysis was done only for the EDM,ISAC, NIAM & REMORA Methodologies out of the 25 which were in the proceedings of the 1st. conference [63].In this study they could not find any observation or evidence which could show the potential ineraction between different features of the Methodologies.

Falkenberg and his group of 9 persons spent a total of one man year in analysing various Methodologies which appeared in the proceedings of the 1st. conference [63]. The group selected the best four Methodologies according to certain criteria, (not stated explicitly), to do a further study and comparision. ACM/PCM, CIAM, ISAC and NIAM were the selected Methodologies. In this study the major principles, concepts, strong & weak points of the Methodologies were studied and suggestions for improvements were also made. Efforts were made to find data Analysis and process Analysis aspects of these four Methodologies. They developed a matrix to show how different concepts and their names have been used by these four Methodologies. The authors did not show any framework and the criteria they used for assesing the Methodologies. Their comments about the good and bad points seems to be with respect to the particular purpose without having wider considerations. The authors have not shown in their paper the criteria for calculating these tables. No argument has been mentioned.

Since this approach was based on very little observation, evidence and limited information, it is difficult to determine adequately and sufficiently how successful these Methodologies are in the real world. How effectively these Methodologies can be used in different environments and the appropriate class of problems. It was not revealed adequately how & to what extent the Methodologies can cover the Span of the System Development Life Cycle. Most of the Methodologies which were selected to do the feature Analysis had a theoretical background with little practical experience.

The third approach was adopted by Maddison and his group during 1983. Efforts were made collectively to find, examine and document the features & the essentials of the Methodologies and to know how these match up to various System development activities. Their objective was "to create from assesment of every methodology descriptions that were similar in content and format" [p-6, 65]. One hundred questions were selected to do a feature Analysis of nine Methodologies with their important variants. Four of those were chosen from the seven selected Methodologies in the first conference organised by IFIP WG 8.1. The group did not mention the criteria for the selection of the Methodologies. Their definition of Methodology is very wide, and is not related with the span of System development tasks. For example, according to their definition JSD is considered as a Methodology. They gave most emphasis to the Analysis aspects in their experiment. Other aspects such as Maintenance, Enhancement, Quality Factors were ignored. Little effort was made to cover Prototyping.

The fourth approach is infeasible because at present there is no Methodology which is commonly used and applicable to all different types of problems & environments. Though many authors make this claim.

The fifth approach at present seems to be unrealistic, because so far software engineering has not been developed to have specific metrics and indices to measure or compare the various feature of the Methodologies quantitatively. Though efforts in this direction are being made.

The sixth approach is the modification of the second and third ones, and is one of the two approaches adopted in this thesis. In this study we are using a common global framework and a set of questions to do the feature Analysis. The feature Analysis is done by the same individual to get commercially unbiased and consistent results, i.e. results free from strong personal belief in support or against a particular Method/Methodology. In this approach the Phases of the global System Development Life Cycle are used as standard tasks for comparison. This comparison is of a relative nature. Due to the problem of terminology, we are covering both Methodologies and certain important Methods in our study. Each Method/Methodology is examined to find how and to what extent it covers the Phases of the Life Cycle. Six Methods/Methodologies have been chosen on the basis of their popularity, significant beneficial effect, contribution to System development tasks and usage in commercial, academic and scientific environments over the past few years. Several questionaires were prepared to be applied to these Methods/Methodologies according to the global framework i.e. the Life Cycle model, to do feature Analysis. Efforts were made to use the terminologies consistently in a way which is very close to that of the IEEE standard and a locally developed glossary. During the study it was realized that most of the questions related to different Phases were common and also the Methodologies do not cover the whole span of the System development tasks. Due to this it was found that the answers to most of the questions were blank. Therefore, it was felt necessary to revise the questionaire. Finally a revised questionaire having 34 questions was selected to apply to the Methods/Methodologies.

The seventh approach is the second one which is applied in this study.

1.2 <u>Objectives</u>

The study of the System development Methods/Methodologies is designed to achieve the following objectives:.

a. to do a comparative study of the Methods/Methodologies and determine their objectives, scope, technical and Management aspects, tools/supports and other major features,

b. to achieve a detailed and comprehensive review of these Methods/Methodologies,

c. to determine the strong and weak points of the Methods/Methodologies,

d. to determine how and to what extent the Methods/Methodologies incorporate the facilities such as documentation, Enhancement, Maintenance, Prototyping, Quality Plan, re-usibility of a System, etc.,

e. to develop a framework which can be used as a standard & systematic way to analyse the Methods/Methodologies,

f. to develop a better, complete and global description of System development, called the System Development Life Cycle,

g. to determine the suitability of the global Life Cycle by examining it analytically to discover how successfully the Methods/Methodologies can be fitted in this Life Cycle,

h. to determine the possibility and means of developing tools which may help in developing economical, better Quality, error free, reliable, re-usable and maintainable software Systems.

1.3 Scope

The following are the scope of this study:

a. Examine in an analytical, critical manner the Methods/Methodologies which are commonly used in commercial, industrial, and scientific environments,

b. Study the Methods/Methodologies which cover and consider the human and Management aspects of the problem during the System development and determine the extent to which these aspects are covered,

c. Study and examine the selected set of Methods/Methodologies on the basis of their referenced materials, published literature, and direct correspondence and arrange discussions with the authors, if needed,

d. Study and compare the terms and their definitions used by the Methods/Methodologies,

e. Develop an un-ambiguous, comprehensive and global set of terminologies and their definitions for possible future use by the Methods/Methodologies. Use these terminilogies during the study of the Methods/Methodologies in order to be consistent,

f. Identify and define clearly the complete span, Phases and tasks which must be done during System development,

g. Recommend the feasible approaches which can be adopted in future to improve the deficiencies of the Methods/Methodologies.

CHAPTER TWO

"System Development Life Cycle"

2.1 Introduction

The term "System Development Life Cycle" is used to identify the main Phases which occur in the development of a System from initial conception to final realization. A glossary of terms used in this thesis is included at the end.

There are several System Development Life Cycles, sometimes called System Development Processes. These Life Cycles are used by various System development Methodologies as an integral part, mostly without definition, e.g., SASD [5,18], LSDM_LBMS [16], NIAM [14], HOS [31] and SADT [11], etc. There are also some System Development Life Cycles defined by Enger N. L. [36], Biggs C.L. [37], Blum B. [38], Ramamoorthy C. V. & Ma Y.N. [39], O'Neill D. [40], Freeman P. [41], and many others. These Life Cycles have several problems in common, for example, not covering the whole span of System development. They are suitable only for certain types of problems and have lengthy Phases which may increase the probablity of committing errors. Their lack of by_pass options for the separate Phases may increase the overall costs for particular projects, after all, unnecessary Phases will cost money. The most critical deficiency is the lack of knowledge about the Requirements of the User/Management, since they do not in general have Phases devoted to this problem.

In real world problems it is a common practice in developing any System to start from some vague idea, some proposal, some 'wants'. At the beginning of a development the actual Requirements are usually incomplete, at least for a short period and possibly for ever in a complex System.

In computer software Systems, it is a well recognized problem that Users of software frequently cannot state clearly and completely what they want until they see a finished System running. Due to this incomplete knowledge of 'wants' and their specification a large number of problems solved by computers are initially ill-defined.

Some explanations for this inconsistency as mentioned above are the present dominating, but inadequate System Development Life Cycles. Mostly these Life Cycles are based on the assumption that the User can express his/her 'Requirements' in a complete and comprehensive way. It has been shown [1], that the most frequently occuring problems arise through (a) incorrectly specified Requirements, (b) inconsistent and incompatible Requirements and (c) unclear Requirements. Past experience reveals that the Requirements of the User/Management cannot be stated fully at the beginning because the User/Management are unable to articulate their wants and expectations completely. Further, they cannot forsee clearly the directions into which further Requirements will develop. "Requirements specification languages are supposed to enable the developers to state their understanding of user's ideas in a form comprehensible to the user.....but the User himself has only very vague ideas of what he/she wants" [2]. The study of present System Analysis and Design Methods/Methodologies reveals that these Methods/Methodologies do not cover the whole span of the System Development Life Cycle in the way we generally solve other problems. Furthermore they do not contain clear definitions of the starting and end points of their Phases. Peter Freeman points out that "repeatedly I encounter life cycles in use that do not fit the 'natural' flow of work that should take place for a given application" [3]. In present Methods/Methodologies there is no check to ensure that the User/Management understands his/her problems and further that what he/she 'wants', really will solve his/her problems. There is no mechanism by which to enquire why there should be a new System, what this System will actually do and how it will do its tasks.

Different Systems Development Life Cycle models given by these Methods/Methodologies have several common features and steps, and use the same terms but for differing concepts, and frequently different terms for the same concepts. These terminologies and freqently used common words are used carelessly and without proper definitions. This causes communication problems especially when two different Methods/Methodologies are used in the same organization. For example, the term **ACTION** is used in ACM/PCM [23] to update an **OPERATION** on an information base. The same term **ACTION** is used in JSD [21] to represent an **EVENT** in which one or more entities participate by performing or suffering the action. The term **EVENT** in LSDM-LBMS [16] is used for **PROCESS**. In SADT [11] the similar term **ACTIVITY**, identified by a verb in a natural language, is used to represent a happening, which may be performed by a computer or people, etc., in a System. In NIAM [14], and ISAC [25], the term **ACTIVITY** is used for an **ACTION** taking time and resources. In MASCOT [22], the term **ACTIVITY** represents a process. The term **ENTITY** in D2S2 [28], LSDM_LBMS [16] and SYSDOC [27] is used for an **OBJECT** which is actually occuring as a concrete thing of interest in a System. The same term in JSD is used for an **OBJECT** in the real world which participates in a time_ordered set of actions i.e. entities perform and suffer actions. In SADT, the term **DATA** is used to represent **OBJECTS or THINGS** working together to perform functions in a System. Furthermore in ACM/PCM, ISAC and NIAM, the term **OBJECT** is used to represent an **ENTITY**. The term **FUNCTION** is used in SYSDOC in a general sense, i.e., to represent any mathematical or business function, where as in JSD the term **FUNCTION** represents an action or set of actions performed by the System and resulting in the production of output. **FUNCTION** in MASCOT represents an action. These Methods/Methodologies start from a Phase which identifies the problems to be solved and give a solution for the problem with the assumption that output will satisfy the Requirements of the User/Management.

The above mentioned problems compelled us to develop a new "System Development Life Cycle", the aim of which is to build a global framework which can help to develop a System in a way which is similar to that which is used to develop Systems for solving other problems. It is an amalgam of all cycles currently found in the literature. All such cycles are incorporated into it on the grounds that they are probably suitable to at least one class of problems. This Cycle consists of a set of Phases, which are generalised. It will be appropriate for all classes of problems and different environments. It is not restricted to any software Design Method or Methodology. This Cycle starts from the project proposal, i.e., the User's 'wants', not from 'Requirements'. There is a difference between 'wants' and 'Requirements' [4]. The 'Requirements' should ideally be targeted towards the 'needs' of the User/Management. The Cycle attempts to identify the true 'wants' and gives a better understanding of the problem. The major purpose of this System Development Cycle is to help the Analyst and the Designer to analyse, Design, implement, and maintain a System in an evolutionary manner. This Cycle therefore covers all aspects of System development such as Project Proposal, Strategy, Analysis, System specifications, Design, Implementation, Transition, and Verification & Validation. The cycle also has facilities to deal with the special cases such as Enhancement, Maintenances, Prototyping and Quality Plan. These cases are optional and can be applied when needed. It is based on the active participation of the User/Management who can help the Analyst in understanding clearly what they (the User/Management) 'want'. This approach provides a conceptual framework for developing a System by using existing powerful and widely used System development Methods/Methodologies and allows the use of established programming Methods such as JSP [42], Top_down, Bottom_up [43, 44], etc.

This Cycle gives clear but flexible Phases which can be bypassed or modified according to the environment and type of problem. In a sense it is a global but at the same time standardized System Development Life Cycle. It offers guide lines and gives some ideas on how to proceed when developing and implementing a System. Any step can be by-passed if found unnecessary. Iteration facilities and Prototyping are regarded as integral parts of the Cycle.

2.2 <u>The Life Cycle</u>

In the following the first figure in the heading is the principal Phase number, the second is the subphase number. The special cases of the Life Cycle start with the letter 'S' followed by the number. The star indicates the Phase name.

1 * Project Proposal Phase

Sub Phases

* Study of Proposal
* Initial Investigation
* Initial Feasibility Study

Before starting any System development task the User/Management will first have to consider how such a System might improve their work and make extensions in their organization. They then prepare a proposal in this regard. In this proposal they may 'want' to have some modifications in the existing System or to have a complete new System which may or may not be computerised. Usually such a proposal is prepared by the Management in consultation with the User. It gives some description of the project and the objectives to be achieved, together with preliminary estimates of manpower and cost involved.

1.1 Study of Proposal

The main objective of this step is to study the proposal given by the Management. There is no doubt that Management and User always face difficulties, as in other fields, in expressing correctly and precisely what they 'want'. In fact in most cases they are not clear themselves what is their exact Requirement. They either propose something ambitiously, thinking 'bigger is better' or some times, underestimate the problem and hence their needs. In this Phase the Analyst will study the proposal, recording possible benefits that can be accomplished with it, taking into consideration the size and working capacity of the organization, budgetary and manpower constraints, annual recurring Systems, operating, Maintenance and training costs which may be involved. The Analyst will try to understand at an initial level what the User/Management actually 'needs'.

The possible outcome of this subphase will be in the shape of some comments or notes recorded by the Analyst for his/her own convenience, better understanding and memory.

1.2 Initial Investigation

Objectives of this subphase are:.

1. to study the most frequent problems likely to occur in the System, in the opinion of

the User/Management,

2. to investigate the proposal and check for correctness, completeness and redundancy of wants,
3. to find out possible benefits or disadvantages that can be obtained from the Proposal,
4. to review 'wants' and seek out actual needs and objectives because it is likely that the proposal will not cover the actual 'needs',
5. to select from the proposal any radical changes that may positively effect the working of the organization and decrease costs.

The Analyst assigned to study the proposal will discuss it with User/Management in a number of meetings to achieve these steps. The output of the Initial Investigation subphase should be a statement postulating actual 'needs' including preliminary identification of cost & benefits.

1.3 Initial Feasibility Study

The basic purpose of this step is to find out possible ways to implement the output of the Initial Investigation step, i.e., the User's/Management's 'needs'. The Analyst has to collect basic information that will be required to implement these 'needs'. The Analyst will then write his/her recommendations about the selection of a particular Analysis Method/Methodology such as SADT [11], SSA [12], PSL/PSA [13], NIAM [14], STRADIS [24] and LSDM_LBMS [16], etc., suitable for this particular problem. The selection of a particular Method or Methodology is not an easy job and, "it is often difficult to determine whether a given methodology applies in a given situation. It is even more difficult to select one methodology from among all those that might be used" [17]. It is not possible to say that a particular Method or Methodology is superior to others in an overall sense even though it is better than others in a particular situation. Therefore, the criteria for the selection of a particular Method or Methodolohy or a combination of them, should depend on the type of problem, the environment and working area and finally the personal experience of the Analyst. Furthermore the Analyst will also write a brief plan expressing estimated cost, manpower required, tentative completion date, equipment required and finally the potential benefits from these changes. The deliverables of this step will be used as a basis for setting detailed objectives and developing strategies for

System development. This report will be submitted to the User/Management for consideration.

2 * Strategy Phase

Strategy means a proposed set of actions clearly designed to achieve certain well defined goals over a period of time.

In the domain of Systems development Strategy means choosing between practices, guidelines, recommmended sets of actions and the ordering of development decisions. In general the Strategy adopted to solve a particular problem, depends on the size and complexity of the problem. In a similar way the Strategy to develop a software System will depend on its size, complextiy and environment. This is because a software System for a complex problem cannot be developed in a similar way to one for simple problems.

In this Phase the output from the Initial Feasibility Study subphase will be used to develop a strategic plan discussing how the Company will proceed with the project. Whether some aspects will be pursued further than others, who will undertake the task and accept the consequences and responsibility of:

1. the changes in personnel, hardware, etc.,
2. the effects of Implementation, Transition and possible delay,
3. the arrangements for formal training and development of new manuals, etc..

This strategic plan should be developed in the light of Managements long term policies.

3 * Analysis Phase

Subphases

* Analysis of Environment
* Requirements Analysis
* Prepare Requirement Specifications
* Check for Consistency
* Search for Existing Available Systems
* Feasibility Analysis & Detailed Proposal

Analysis means to decompose something (which is under study) into its component parts for identification, examination and interpretation. The term Systems Analysis is used to break, an old System (if it exists), or an idea for a new one into its components in an attempt to reveal and examine how those components work and interact with each other in order to accomplish the purpose. In general it is a job whose purpose is to show 'what is in' the System and 'how it works' within the environment of the User/Management. "In the specific domain of computer system development, analysis refers to the study of some business or application area, usually leading to the specification of a new system" [5]. The term 'Analysis' is used in this System Development Life Cycle in a different way. It is used for the purpose of Systematic study and examination of 'needs', (obtained in Phase 1), in order to find, determine and identify the causes of the problems and the Requirements of the User/Management. The results of this Analysis are represented in detailed proposal(s) together with the feasibility report(s) for developing the System Specifications. In these proposals the Analyst develops a definition of the Requirements and Requirements Specification, important parameters, and the environment in which the System will work. Emphasis should be given to the Analysis Phase in order to get a better understanding of User's/Management's Requirements and expectations.

3.1 Analysis of Environment

In general, no Designer designs a System badly by choice, but at present many Systems which are working in different environments are either working badly or are total failures. One of the possible reasons for that is that both Analyst and Designer are unaware of the problem area and its environment.

During the tasks of System development the Analyst often does not have much information about the User and the environment for which the System will be designed. Hence an outcome of the development tasks may be something over which the User/ Management disagree with the Analyst, resulting in dissatisfaction and frustration. An Analyst who does not have a thorough understanding of the environment, working area and its terminology cannot communicate effectively with the User/Management and convince them if required. Brown G.L. mentiones that "A high quality of system design will be very difficult to produce unless systems designers have an adequate understanding of the industry in which the organization does business .... (and the) technical environment in which the system must be implemented" [6]. Unfortunately in present existing cycles too little emphasis is given to the importance of collecting information about the environment in which the System will be working.

Objectives of this subphase are:

1. to collect and understand information about the environment in which the System will work,
2. to know about the application area for which the System will be designed,
3. to develop a dictionary of all different terms, concepts, notations and definitions which are used in the establishment, in order to avoid misinterpretation of words and communication problems,
4. to investigate factors which may effect the System environment, i.e., factors which are responsible for the occurrence of incorrect data, creating inefficient processes or functions, and
5. to investigate factors which cause the System to be less portable, create security and integrity problems, etc..

The output of this subphase is a small report showing adequate information about the environment and working area for which the System is being designed. This report will be used by the Analyst in subphase 3.2, i.e, Requirements Analysis.

3.2 Requirements Analysis

"Requirements analysis deals with the difficulties of understanding the problem and of communicating that understanding among the concerned individuals and organizations" [7]. In short, Requirement Analysis encompasses all aspects of the System and yields the results from which Requirements Specification can be build. Its main objectives are:

1. to decide clearly whether or not those identified needs will actually be required,
2. to collect further information about 'needs' and formulate its structure,
3. to consider the social, legal, security, privacy, managerial, and, financial impacts which may come as a result of fulfilling the above 'needs'. These aspects may impose additional Requirements on the System.
4. to consider the environment where the System will work,
5. to communicate all the above information

to User/Management.

A part of this work can be done best by using the "Participative approach to System Design Method" [8] [9], which gives emphasis to the need for meeting the human needs of staff when designing computer Systems. This Method is supported here because the User is likely to be the most appropriate person inside the organization, who may know something about the 'needs' and predict any further changes in these User 'needs' which may possibly occur during the tasks of System development. This is because the 'needs', may not be stable throughout the entire System development tasks. The tasks of Requirements Analysis performed in the above way can help the User, Management and Analyst to learn from each other and thus to write a better Analysis report.

The product of Requirements Analysis must be a clear statement of the User's/Management's Requirements some sound results from which the Requirements Specification can be built. "The evidence is overwhelming in industry regarding the dramatic consequences of incorrect or inefficient problem definition work. The causes are misunderstanding of the problem and insufficient analysis to decompose the problem" [10]. Some essential properties of the Requirements must be:

Completeness,

Consistency,

Unambiguity and

Modifiability.

3.3 Prepare Requirements Specification

It is difficult but desirable to have a clear definition of the Requirements Specification. The problem in formulating a clear definition is that different authors have different concepts about 'Requirements' and 'Specification' and hence of 'Requirements Specification'. We are proposing a definition which is a general one. To specify is to state particularly and precisely. Hence a specification is a particular and precise statement of something. A Requirement is something which is deemed to be needed. A Requirements Specification is therefore a particular and precise statement of those things which are deemed to be needed. Note that this perception of needs is from the viewpoint of the User, their Management and the environment.

The main objectives in preparing a Requirements Specification are:.

1. to define clearly what the System or part of it must do in accordance with agreed Requirements, i.e., the output of the subphase for Requirements Analysis,
2. to help in transforming the Requirements into a System specifications by providing a well defined starting point,
3. to act as an initial model which can

exhibit, as much as possible, relevant information for forming a Design,

4. to specify initially the data, processes or functions which might be required in the System,
5. to act as a basis for System assessment.

The preparation of a Requirements Specification is a complex and time consuming job. Its success depends to a great extent on the Quality of the Requirements Analysis. This job needs high Quality, intellectual, creative and experienced Analysts.

The output of Requirements Specification must be expressed in a clear and comprehensive report. This report will be used further as a basis to prepare detailed proposals for the new System and can either be prepared by using a specification language or a natural language supported by diagrams.

A Requirements Specification having the following properties can be considerd to be of high Quality.

<u>Feasibility</u>

Requirements Specification which are developed in such a way that they are capable of being used successfuly and are according to the agreed needs and constraints mentioned in Phase 1, are said to be feasible. It is difficult but important to develop such Requirements Specification. Lack of feasibility may cause serious Implementation problems and hence increases cost and frustration.

Understandability

Requirements Specification which are developed in such a manner that they are clear, unambigious and difficult to misinterpret are said to be understandable. During the tasks of System development some Requirements Specifications are ignored intentionally or unintentionally by the Designer because of these problems. It is, therefore, necessary to avoid errors and omissions in the Requirements Specification and to develop them in such a manner that they are understandable. The understandability of Requirements Specification can be obtained by imposing constraints on the style of writing in natural language and using diagramatic help.

Reliability

A Requirements Specification having a high probability of being complete, correct, and consistent and which can lead towards successful development and Implementation of a software System over a period and under predefined conditions, is called a Reliable Requirements Specification. Unreliable Requirements

Specification leads towards uncertainty as to whether or not the System can be successfully implemented. To get a reliable specification it is essential to verify consistency between Requirements and Requirements Specification. This can be done, as mentioned earlier, by using the participative Method of Systems Design in which the Requirements Specification is shown to the User/Management to avoid complications, chances of misunderstanding and to reduce discrepancies, before Implementation.

Evaluation

The Requirements Specification should reflect the Requirements of the User/Management which may not be static and may change whenever they want. These changes may be due to improvement, elaboration, specialisation and generalization. It is, therefore, necessary to develop a Requirements Specification which is evolvable. In our System Development Life Cycle any changes or modifications should compel the Analyst/Designer to go to the start of Phase 3 and repeat the same excercise. It will be easier to do this as the previous skeleton or model for the System can be re-used.

Maintainability

Inevitably errors will be introduced in the final product which have not been trapped during the previous stages. If it is important to the User that the System be corrected then the ability to maintain the System will be a Requirement. Transporting the System to other machines may also be a part of this Requirement. The criteria by means of which this Requirement can be enforced must be considered and Implementation anticipated.

3.4 Check for consistency

By consistency we mean to examine critically in order to locate any contradictions in the Requirements Specification which may be due to:

1. a difference between the Requirements Specification and what the User/Management 'intended',
2. modification of the Requirements Specification, which may happen from time to time,
3. contradiction and incompatibilities between Requirements,
4. missing components, undefined entries, etc..

After every modification care must be taken to ensure that consistency is maintained. For these reasons it is essential to have frequent consistency checks. These checks should be made before moving to further steps of the System Development Life Cycle, especially before the

Search for Existing Available Systems and writing Feasibility Analysis and Detailed Proposals. Any error at this level may cause many problems in future steps. To eliminate contradiction at this stage is not an easy job and becomes more difficult when the System being modelled gets bigger and more detailed. This problem can be greatly simplified if a detailed cross_reference is available.

3.5 Search for Existing Available Systems

Shortage of skilled and experienced Analyst/Designers and their cost of employment is becoming a serious administrative problem. For example, it is estimated that up to 90 percent of the data processing intellectual effort in a large corporation is devoted to developing and Maintenance of software [45]. "It is commonly recognized that software reusability could provide powerful leverage for reducing future software development cost" [55]. Therefore this problem can be eased by making use of available proven Systems from the System market. This involves detailed searching of software System banks/catalogues to pick up a System suited to the Requirements Specification, or otherwise modify one of the Systems which is similar to the required one and involves a minimal degree of changes. In such an approach the Analyst has to make sure that the System which is being selected is sufficiently free from bugs, has sufficient good documentation and is acceptably User friendly etc. This information can be obtained in a number of ways, e.g. by submitting a number of questions to various establishments using the chosen System. After this selection the Analyst can move directly to the Implementation step by-passing all intermediate steps of the cycle which are discussed below. Before this the Analyst has to satisfy the User/Management that the Implementation is consistent with the Requirements Specification. Purchasing Systems in the above manner will cost less money and time than creating it by oneself because the development cost of a System available commercially is spread over a number of purchasers.

It may be that a System can be found which is close to the required System. In this case commercial consideration may force the omission of these missing Requirements. Clearly some considerable thought must be given to finding these missing Requirements and then studying the implications of these omissions.

3.6 Feasibility Analysis and Detailed Proposals

Feasibility Analysis means studying how the Requirements Specification can be implemented within the given constraints such as technical, operational and economic factors. It is useless and painful to Design a System which cannot be implemented under given constraints. In this subphase the Analyst will prepare a detailed report, the purpose of which is to give Management:

1. a detailed review of the Requirements and Requirements Specification and surety that there are no redundant factors and that its Implementation is feasible,
2. the expected Implementation, running and Maintainence costs of the new System,
3. a detailed plan with which to develop, implement, and test the new System,
4. estimates about completion time, staff needed and hardware required,
5. cost benefit Analysis for the new System,
6. information about essential changes in policies, environment, status of personnel, organizational changes and effects thereof.

The Analyst should select first those Requirements which may cause major changes in the existing System (if there is one), and are relatively expensive to implement. On the basis of this selection the Analyst will recommend whether it is advisable to implement those Requirements or not. This report should also justify the Implementation of the rest of the Requirements for the new System, keeping under consideration the Implementation, running, and Maintenance costs against possible benefits which Management may get from the new System.

The Analyst should inform the Management in the report about the expected results which may come after implementing the new System, and possible effects if the old System is continued.

Systems which are developed purely to achieve certain technical objectives may have unpredictable human consequences, because technical decisions may cause changes in policies, Strategy, environment, status of personnel, and organization. These changes may result in serious industrial relations problems. It is, therefore, necessary that the Analyst in the Feasibility Analysis and Detailed Proposals should avoid chances of developing human aggravation and provide a way to gain the User's confidence.

Sometimes changes may occur during the development tasks which cannot be ignored. These changes may be due to the time factor and a better understanding of the Requirements, or earlier communication failures and their correction. It is recommended to submit more than one complete proposal with the Analyst's recommendations, and conclusions, giving summarized statistics about, cost, time, etc. and comparisons between the present and the new System and the possible benefits thereof. These proposals should be discussed, before any move, by User/Management or a committee on their behalf. They will decide accordingly which proposal should be accepted and approved for further progress. However, if User/Management or their committee is unable to decide and does not agree with any proposal then it will be essential to repeat the Cycle, do all the excercises again, and submit new proposals or otherwise terminate the project.

4 * System Specifications Phase 4

System specifications should state clearly and unambiguously, what a System must do. It is a precise statement of the Requirements in a form which is meaningful to the Designer. There are three main description Methods for System specifications. Firstly the use of formal notation Methods, such as VDM [30], HOS [31], and AFFIRM [35] etc.. Secondly Methods like PSL/PSA [13] and SREM [15], etc., which emphasize Management and documentation aspects. These previous Methods are not widely used in industry. Thirdly, and most commonly, there are specifications expressed in natural language.

The System Specifications Phase represents an interface between Analysis and Design Phases. This Phase separates completely the Design work from the Analysis and refers to that information which must be delivered from the Analysis to the Design Phase. It converts the terminology and pragmatics of the application area into the terminology and pragmatics of computing. For example a User's Requirement that a System should be reliable to a given extent must be transferred into constraints which are relevant to computing, such as appropriate values for reliability metrics. Another example would be that a Requirement for completion of a document for a particular task would lead to a precise definition of the contents of that document in terms of the inputs to the computer System. This Phase separates the User's/Management's and

Analyst's view of the problem, and its solution, from the Designer. Without such an interface it will be difficult to know how to represent the output of the Analysis results technically. Further such an interface helps to determine, up to a reasonable extent, that proposed changes will not have any adverse effect on the present System (if there is one). This interface will also help in providing flexibility in System Design, enabling the Designer to execute easily the changes which may occur from time to time. System Specifications can be considered as a basic source from which alternate Design decisions can be made.

The main purposes of System Specifications are:

1. to express in precise and consistent form what the System will do, according to the User's/Management's and Analyst's agreed proposal,
2. to define the objectives and constraints which the System must satisfy, e.g., System boundaries, System interface, etc.,
3. to specify details about processes, functions and operations of a System,
4. to specify provisions for error handling and recovery thereof,
5. for use as a standard against which an Implementation can be verified,
6. for use later to support System

Maintenance,

7. in some cases for use in building a prototype.

The Quality of a System depends to a great extent on the way System Specifications are developed. In this Life Cycle, System Specifications are developed from a selected detailed proposal, i.e., output of Analysis Phase-3. In the System Specifications Phase the Analyst/Designer elaborates the proposal into a set of different clear steps which are used as a basis for System Design. This elaboration can be based either on functions, processes or data, and some times on both. It may lead towards one or more concepts of System Design such as Structured Design [18], Top-down Method, Bottom-up Method [43, 44], JSD [21], MASCOT [22], etc., as the case may be, according to the type of problem. It is, therefore, recommended that System Specifications should not be described in accordance with a particular Design Method/Methodology. Again here at this point we are not ranking one Design Method or Methodology above an other. We consider System Specifications as the foundation of Design, therefore, it is essential that User and Management must agree and confirm that these System Specifications represent their Requirements and serve their purpose. This will help and encourage the Designer in making further steps.

Many Implementation and Maintenance problems are due to incomplete specifications. Incompleteness in specifications implies that the Analysis work and its output are incomplete and, therefore, may require a return to the Analysis Phase (Phase-3). Paying sufficient attention during the development of System Specifications can result in reducing Maintenance cost of the System.

5 * Design Phase

Sub Phases

* Planning and Decision
* Conceptual Design
* Physical Design

"Design is the process of transforming what has to be done into a means of doing it" [18]. In this Phase emphasis will be given to the application of the System Specifications, rather than understanding the needs and Requirements of the User/Management. "Design is concerned with what might be in the future if the design be implemented, and hence is concerned with what is not yet or does not yet exist" [19]. In this Life Cycle the term Design is used both as a verb and a noun. As a verb it is used as conceiving & planning by using an agreed proposal(s) and its specifications as input and converting the plan into an unambiguous executable model, called a System frame or System skeleton to perform its purpose(s) i.e. function(s). As a noun it is used as a tool set which accepts the human proposal(s) and its specifications as input and can develop an unambiguous executable System frame or System skeleton to provide the information to the User/Management and serve his purpose(s) i.e. function(s). The Design Phase starts after finalising the System Specifications which are in fact a plan for the detailed proposal expressed in technical terms. This Phase must preceed and not be confused with Implementation, i.e.

its scope is up to Implementation point. The Design tasks, if performed carefully, converts this plan into an unambiguous model, (although there may be more than one model). It involves devising ways which lead to the attainment of the required goals, i.e., to solve the problem in a way which is according to the proposal and its specifications and satisfies the User/Management. Usually experienced Designers have certain assumptions, beliefs and tricks and they develop a Design with the help of this known-how, i.e., their past experiences. They use these as a theory consciously or unconsciously. Such assumptions, beliefs and tricks are usually correct and worth using, if the resulting Design is passed through well prepared testbeds. The recent study of System development Methods/Methodologies revealed that in general there is no Method/Methodology or tool which is suitable for building a System for all different types of problems and environments and is capable of satisfying or likely to satisfy different User's/Management's needs, Requirements and corresponding specifications. Any Design problem can be solved in an infinite number of ways. This number eventually is reduced by deciding to adopt a single or a set of particular Methods or Methodologies. There are several Design Methods or Methodologies suitable for different fields and environments, such as Structured Design [20], SADT [11], JSD [21], MASCOT [22], ACM/PCM [23], ISAC [25], LSDM-LBMS [16] and STRADIS [24], etc., each one having many drawbacks. These Design Methods or Methodologies, can be classified according to four distinct main approaches, i.e.

1. Building a System by giving emphasis to functions and processes. This is called functionally or process based Design such as ISAC, NIAM, SASD, etc..
2. Designing a System by giving emphasis to the flow of data to be used, i.e., describing the System in terms of data. This is called a data driven System, such as SYSDOC, D2S2, LSDM-LBMS, etc..
3. Designing a System by developing a model of the real world. This is done without initially mentioning the functions to be performed by the System, but introducing these functions explicitly at a later stage, by using verbs in the specification. The JSD Method is of this type.
4. Designing a System based on the States of entities and processes or their behaviour and changes in any combination within the environment. Designs based on SDL [32, 33] are of this type.

Criteria for the choice of a particular Method or Methodology, as in the Analysis Phase, should be based in the Feasibility Analysis and the Detailed proposal. To obtain a good Design the following are some general guidelines and recommendations; but of course as mentioned above, the final selection must be based on considerations of the type of problem under study.

1. Similar to the Analysis Phase develop a dictionary of all the different terms, concepts, notations, and definitions, which are to be the in the System in order to avoid misinterpration of words and communication problems.
2. Decompose the overall problem and its specifications successively into smaller and smaller subproblems, possibly called modules or subsystems. This decomposition continues till a point is reached from which the Design looks relatively easy. The criteria for this decomposition depends on the type of problem, environment and experience of the Designer.
3. The modules or subsystems must be able to be conveniently and coherently connected.
4. Each major module or subsystem must be further investigated with a view to finding out its main functions, processes, entities, actions, and their relationships, etc..
5. Each module or subsystem must be tested to make sure it is performing its functions and processes up to a satisfactory level.

This needs the development of suitable test data having acceptance criteria and possibly mentioning the time dependence between modules or subsystems.

6. Each module or subsystem should be protected against the possibility of incorrect input data. This needs the development of certain constraints to prevent incorrect input.
7. Modules or subsystems must be easy to integrate, manage and perceive, i.e., which can easily be implemented, maintained and amended.

As the Design is decomposed it may be necessary to use the Life Cycle recursively. This is because further Requirements relevent to a subsystem may need to be determined or clarified. In this case many of the subphases need not be performed.

The detailed subphases for Design are as follows.

5.1 Planning and Decision

"Design involves both making decisions about what precisely a system will do and then planning an overall structure for the software which enables it to perform its task" [26]. The Planning and Decision tasks involve using the System Specifications to develop the System. In this subphase crucial decisions about the basic structure of the System Design and how to convert them into a series of hierarchical steps are made. At this stage, great care is needed as these decisions have great impact on the final System Design and can have consequential effects for the later Maintenance Phase. This job needs consideration and it is probably of an iterative nature.

The Designer must establish, in consultation with the Analyst and the User/Management, clear definitions, main functions and general processes of the System, operating parameters, nature of data to be stored, and patterns of output. The Designer must prepare a list of instructions and formulae, if there are any, depicting how to solve the problem. Clear definitions of functions and the general processes of the System will help the project team in understanding and deciding which functions and process are to be taken and which are to be by-passed. It will further help in making decisions about which tool, Method or Methodology should be implemented.

5.2 Conceptual Design

The Conceptual Design or visualized model means an appropriate image of the actual Design, its information flows, processes network, entities, functions, modules, subsystems, their relationships, and interactions in the

Designer's mind.

In this subphase the Designer will make certain basic assumptions about the Design to be built based on the decisions and planning made in the previous subphase, the environment, nature of the selected proposal and System Specifications thereof, developed in Phase_4, and of course his/her experience. The Designer will produce ideas for creating modules, subsystems, databases, etc., for example:

1. The Designer can assume that it will be better to develop a conceptual model of the real world, based on the results of Analysis and System Specifications Phases; and later introduce functions. Thus the Designer uses the JSD Method with the essential required knowledge about the Systems environment, Analysis results, and System Specifications, i.e., some extra items which are missing in the JSD Method.
2. The Designer can build a Conceptual Design or model about entities types, relationship types and data element types using the Analysis results and System Specifications e.g., using the SYSDOC.
3. For a Real time System the Designer can make a Conceptual Design based on the Analysis results and the System Specifications showing that changes in the

computer System will occur in the same sequence as changes in the environment, e.g., SDL Method.

4. A Conceptual Design of a System consisting of object and activity classes and rules can be developed by using the information obtained from the Analysis results and the System Specifications. The description of this Conceptual System is called an abstraction System by NIAM, and it can be expressed by using a conceptual grammer.

5. If the Structured Design approach is followed, then a Conceptual Design can be built by developing concepts about data, data structure, data flow diagrams, a data dictionary, hierarchies of modules, etc. based on the Analysis results and the System Specification.

The output of Conceptual Design should be documented properly after consulting with the Analyst and the User/Management for further use in developing Physical Design.

5.3 Physical Design

By Physical Design we mean designing details of data, data format, data structure, data base, data dictionary, functions, processes, modules, subsystems, etc., according to the selected Design Method or Methodology which can be represented and implemented physically, (like actual things) at a particular time in a System.

In Physical Design each module or subsystem will be physically or actually identified along with its interfaces with other modules or subsystems.

To develop a Physical Design in this Life Cycle the output of Conceptual Design will be used. In this subphase modules or subsystems, data dictionaries, data bases, etc. conceived in earlier subphases will be developed physically in detail.

The Physical structure of data, data dictionaries, modules or subsystems etc., for example, can be described by charts, diagrams, tables, and coding formats. This involves the development of:

1. dataflow diagrams,
2. structure charts, decision tables, etc.,
3. modules or subsystems as a part of a hierarchy and showing their points of entry and exit,
4. input details, and the layout of, for example, the data format, the data structures, the input variables, and their points of entrance into the module or

subsystem etc.,

5. output details and their layout e.g. output, output structure of files or databases, their point of exit from modules or subsystems and conditions thereof, format of error messages, and physical source of output etc.,
6. Details about file structures, and the procedures to retrieve and update, indicating which files are to be permanently kept.

The Physical structure of processes, procedures, activities, and functions can be described by using structured English, pseudocode, machine processable language, diagrams, charts etc.

The Physical Design of User interface can be developed from User options, clerical routines, available or proposed hardware. This interface will assist the User/Management in the selection of the report option.

The output of the Physical Design subphase will be a Design in which each System module or subsystem will be physically available along with all its interfaces to other modules or subsystems, and will be matched.

It is important to mention that the tasks of developing the Physical Design like other subphases is of an iterative nature, and needs frequent consultation with the Analyst and the User/Management. The output of this subphase will be available for the Implementation Phase.

The details about modules or subsystems, their control, dataflow, output layout, different constraints etc., elaborated in the different Design subphases, will be integrated to form a frame or skeleton of a System to be used in the Implementation Phase.

On the basis of this frame or skeleton the Analyst/Designer will be able to estimate the program development, installation, and Implementation cost of the System. This cost estimate will be submitted, along with the Design frame, to the User/Management for their approval prior to any further move.

6 * Implementation Phase

The IEEE defines Implementation as, (1) "A realization of an abstraction in more concrete terms; in particular, in terms of hardware, software, or both". (2) "A machine executable form of a program, or a form of a program that can be translated automatically to machine executable form". (3) "The process of translating a design into code and debugging the code." They also define the Implementation Phase as, "The period of time in the software life cycle during which a software product is created from design documentation and debugged." [34]

Implementation means the tasks which are concerned with giving practical effect to the Design or System skeleton: the manipulation & translation into code of a software System frame developed earlier in the Design Phase, to fit it to the hardware and also in some cases to the other software. It is also concerned with debugging the code, if necessary.

By Implementation Phase means the period of time in the System Life Cycle, during which the Implementation tasks expressed below in this section, are performed. The value of this frame or skeleton can decrease if a systematic approach is not followed.

There are a number of ways or strategies to implement that frame or skeleton such as JSP [42], Top-down [43], Bottom-up [44], Structured [50,51], Stepwise refinement [52], and Modular [53], Programming approaches. Criteria for using a particular approach should depend on the type of the problem, the Design Method used earlier and the environment.

For example:

1. a Strategy to implement a System based on MASCOT will be different from that of SDL,
2. for the processes having sequential input and output data streams JSD might be used to develop the Design frame. So it will be appropriate to use JSP for developing the Program,
3. for a Design developed based on SASD Methodology, Structured Programming might be suitable, etc..

This frame or skeleton is converted into the detailed Program statements by coding into a suitable Programming language and developing it's documentation. The choice of proper Programming language and documentation technique needs due consideration [46]. Because some languages and documentation techniques are not suitable for certain class of problems and machines. Selection of improper Programming language and Programming Method can make Implentation difficult.

The Implementation task in this Cycle varies from System to System, on the Design Method or Methodology adopted during the Design Phase, and the hardware which is being used, for example:

1. In JSD where most of the Design decisions are taken at Implementation level [21, P-25], it's Implementation task will be different than if the same problem is developed by using SASD.
2. Implementation of a real time System such as an air traffic control System where an immediate response of an event is needed, will be different from that of a simple accounting System.
3. Due to rapid development in hardware and it's low cost, it is now possible to implement in hardware certain functions which generally used to be Implemented by using software

Briefly the Implementation tasks include the following:

1. preparation of an Implementation schedule in consultation with the User/Management,
2. development of accurate & reliable computer programs based on the System frame or System skeleton from the Design Phase.
3. scheduling of modules, subsystems, processes and their mechanisms for

invoking and suspension,

4. checking that the required hardware and software (in some cases) are available and are in working condition,
5. development of operating procedures, manuals, and documentation in a clear and understandable way,
6. training of the end User/Management in the use of the System and the interpretation of the output etc.,
7. helping and guiding the User/Management in developing the organisation and administrative structure of the data processing centre,
8. ensuring that all the required changes have been implemented and nothing is missing or is ambiguous.

The success of Implementation depends on how well the frame or skeleton has been developed. A good frame can save much time and money, which is usually spent on debugging etc.. It will be better to implement the System, initially on a pilot study basis. This Strategy will help in evaluating the deficiencies, if there are any, and making the changes before final acceptance, if necessary.

During Implementation effort must be made to ensure that the System has been implemented according to the Design and can perform it's objectives set in the earlier Phases. The User/Management should attest that the System has been implemented properly and it's performance is satisfactory to the extent that no serious problems are occuring.

During this Phase V & V Phase must be applied to complete the System testing up to a reasonable level.

7 * Transition Phase

Transition tasks are defined in the Life Cycle as the evolution and change from one System to another and hence occur only in situations where both exist. Transition tasks are concerned with how the new System should replace the old one and deals with all the problems which may occur during the tasks of switchover from the old to the new System, such as:.

1. switchover tasks may effect the existing working System due to changes in the normal routine.
2. deciding which function or procedure should be introduced first.
3. having an interface between the new and the old System.
4. training the User/Management and also increasing their motivation.
5. effects on the organizational structure which may create serious adverse effects.

This Phase may involve the following:

1. User/Management training,
2. Data Conversion,
3. Parallel Operation,
4. Operational Documentation,
5. Acceptance.

In this task efforts must be made to train and guide the concerned staff and increase their motivation. In certain cases their work and performance should be examined carefully during the Transition Period.

Existing data, datafiles, data structures, entities, etc. must be copied and converted according to the input Requirements of the new System. It must be decided which existing files will be needed in the new System. This task could have been in parallel to the previous Phases. This transformation may involve extra cost especially when the organisation is decentralised.

Ideally the new and old Systems should run in parallel for the purpose of experimentation, observation and training the staff. This should be continued until the new System is accepted formally by the User/Management.

In some cases the results of the Transition Phase indicate that the System may require Maintenance or Enhancement. This may be required to eliminate errors, omissions and improvement in the efficiency and performance of the new System. This may force the Analyst/Designer to go back to the approproate point in the Life Cycle and re-do the required things again. This can easily be done by activation of any suitable special cases of the Life Cycle.

The Transition tasks needs close involvement by trained and experienced staff for the supervision and examination of the performance critically. Before accepting the new System the staff concerned should wait for consistent results from the operation of the new System.

This Phase differs from environment to environment and System to System and is of problem oriented. In this Phase certain possible tasks may start in parallel to the Phases after the Analysis. For example:

1. After developing specifications about the processes, functions etc. the Tarnsition Phase may be started by elaborating the details about the processes or functions to be developed.
2. If the Design decision is that the JSD will be used then the Transition tasks may be started by making efforts to develop the model of the real world.
3. If the Design decision is that a state base System will be developed then details about these states must be made.
4. Efforts to develop the data dictionary, terms, notations, definitions etc. can be started much earlier during the Phases after the System Analysis.

etc..

V * Verification & Validation Phase.

Verification is defined variously by IEEE as:.

1. "The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous Phase."
2. "Formal proof of program correctness."
3. "The act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether or not items, processes, services or documents conform to the specified requirements" [34].

Verification implies to the establishment the correctness of something by comparing it against some standard. For example to compare the developed System with its Requirements, System Specifications, etc.

Validation is defined by IEEE as:

"the process of evaluating software at the end of the software development process to insure compliance with software requirements" [34].

Thus Validation ensures that the output of a Phase complies with a standard such as the Requirements, the Detailed Proposals, the System Specifications, etc..

The purpose of a V & V Phase in this Life Cycle is to:

1. guide the Analyst/Designer and the User/Management in making decisions about their next move, which may be to go ahead or abandon the project,
2. satisfy that the System Specifications are developed according to the result of the Analysis Phase and serve its purpose,
3. satisfy oneself and the User/Management that the System Design is developed according to the Requirements and the System Specifications,
4. check the output of the Design Phase against the basic rules of the selected Design Method or Methodology, such as rules about input/output data structure, criteria to build modules or subsystems, their connections and calling or exit procedures, etc, and
5. ensure that its result satisfies the test data.

The V & V Phase may be activated at any time and in any Phase of the Life Cycle, if needed. Specifically it is activated after having:

1. Requirements Specification,

2. Detailed Proposals,
3. System Specifications,
4. Design output,
5. Implementation,
6. Prototyping.

For example.

1. The V & V Phase should be called during the Analysis Phase to ensure that the Requirements Specification and the Detailed proposal(2) have been developed properly, because any error at this Phase can create major effects in terms of complexity, time, and cost.
2. The V & V Phase should be used during the Design Phase to ensure that the modules or subsystems are complete, cohesive and minimaly coupled. That the combined effect of the modules or subsystem, data flow, control flow etc. is according to the Requirements & System Specifications. Furthermore it must be documented that the software is feasible to implement and can serve the purpose and the environment for which it was designed.
3. The V & V Phase must also be applied during Implementation to ensure that the Program is an appropriate representation of the

Design output and no unapproved changes were introduced by the Programmer during the Phase. It will be better to use another V & V Phase after integration of modules or subsystems to check their combined effect.

4. The V & V Phase can be used for detecting unperformed Requirements and mistakes, which later can be traced back to find the source of error.

The V & V Phase can be achieved by,

1. Testing or Auditing,
2. Inspecting & Reviewing,
3. Implementing & Running the Phase.

Tasks such as testing the output of a Phase, must be based on testing against certain pre-selected factors and constraints to ensure that the Requirements and the System Specifications have been implemented properly.

The goals of the V & V Phase can be achieved by inspecting and reviewing the selected factors which according to the User/Management may effect the performance of the System. There are several types of human inspection available including walkthrough, review and inspection [70].

Many of the V & V activities for any Phase can also be achieved by running and implementing the System or by using a Prototype variant of the Life Cycle.

Great care must be taken in preparing the list of the important factors and constraints which might be used for the above purposes. This list must depict the sequence and level of extent up to which the V & V Phase will be done. The output of the above tasks must be documented for possible future use.

If the User/Management or anyone concerned is not happy with the output of the Validation Phase, it means that perhaps:

1. User/Management have changed their mind about what they 'want'.
2. The Analysis was not done properly.
3. The System Specifications were not developed carefully.
4. The selected detailed proposal, the basis of System Specifications does not reflect the Requirements of User/Management or is not suitable for that particular problem and the environment.
5. The Design tasks were not done properly.
6. The selection of the Design Method or Methodology is not appropriate and cannot solve the problem properly, etc..

Hence due to any of the above reasons it will become necessary to go back to the appropriate Phase and do the whole excercise again using the recursive property of the cycle.

This re-useability of a V & V Phase in the Life Cycle can certainly increase the Quality of the output for the particular Phase in which it is being used and as a direct result the overall Quality of the System.

Special Cases

The following are the special cases of the Life Cycle.

S1 * Enhancement

S2 * Maintenance

S3 * Prototyping

S4 * Quality Plan

S1 * Enhancement

This is a special case of the Life Cycle. Enhancement means changing the System due to the development of additional or changed Requirements. This task is dependent to some extent on; how the System being enhanced has been developed; the Method or Methodology adopted, and the System environment.

The reason for Enhancement may be the User/Management want to have more facilities, increase performance, improved Quality Factors and other attributes of the System. All these require changes in the Requirements.

When the complete Requirements are finalised then it is known that a System which nearly satisfies these new Requirements already exists (it is the present System). The remainder of the Life Cycle therefore, involves changing the existing System Specifications, Design, and Implementation to reflect the new Requirements.

Usually the Enhancement tasks starts after completion of the System. To perform these tasks the Analyst/Designer have to proceed in a similar way as in the main Cycle starting from the User's Proposal until the last Phase of the Life Cycle, bypassing those Phases which are not necessary.

The difference between Maintenance and Enhancement at times is small. Correcting errors can often involve a complete pass through the product of the whole Life Cycle in order to discover the corrections to these errors.

S2 * Maintenance

The IEEE defines Maintenance as:.

1. "Modification of a software product after delivery to correct faults,
2. Modification of a software product after delivery to correct faults, to improve performance, or other attributes, or to adopt the product to a changed environment" [34].

Where as:.

1. Adaptive Maintenance is defined as "maintenance performed to make a software product usable in a changed environment".
2. Corrective Maintenance is defined as "maintenance performed specially to overcome existing faults".
3. Perfective Maintenance is defined as "maintenance performed to improve performance, maintainability, or other software attributes" [34].

"In the software context this (maintenance) means the removal of errors found after acceptance (debugging), the improvement of performance by revised design and implementation, and the development and modification of facilities as operational experience is gained or the requirement changes" [22].

JSD uses the word Maintenance to denote the activity of changing the system to meet these changes (i. e. changes in functional requirements, changes to the real world, etc.) in specification [21, p.352].

In this Life Cycle Maintenance means the changes incorporated into the System at any point from Requirements Specification in the Life Cycle onwards, to:-

1. rectify faults, bugs etc. which may appear from time to time,
2. make the System workable for a particular environment and hardware.

System Maintenance is very often a reccuring problem. This problem effects both the User/Management and the Analyst/Designer. There are several well known reasons which can force changes and improvements. The most obvious ones are due to basic Analysis and Design errors. These changes may be due to the addition of certain functions, processes or modules, to introduce more facilities and Quality Factors. Such changes sometimes may happen during the tasks of System development. In such cases the Analyst/Designer has to modify the proposal or change the Design frame or skeleton. However if the User/Management want to make the changes to fix defects in the existing working System, subsystem, module, etc., such changes become very difficult to incorporate, may conflict with efficiency and reduce the life and utility of the System.

Any attempt to incorporate the changes and to rectify the defects locally without invoking the essential Phases of the Life Cycle may repair the local and obvious defects by creating other defects. A possible reason for this may be that these modifications usually have ripple effects which may result in introducing other defects. Brooks [54] states that a computer program modification carries with it 20 to 50 % chance of introducing a second error. These effects become worse, if the System under Maintenance was not developed by using an appropriate Analysis and Design Method or Methodology.

There is also an another problem. These existing Methods or Methodologies generally have no specific or universal measure or metric for Maintainability, or something similar to it, which can help to determine the extent to which the tasks of Systems modification, correction, and debugging should be continued. There is no specific or universal criteria or standard which can measure the efforts required to locate and debug an error or can determine when it is essential to rebuild a System. There is no clear indicator to show when System Maintenance is not feasible. All these usually depend on personal decisions.

There may be several reasons which can cause Maintenance tasks, such as:.

1. Changes in the Requirements Specification.
2. Changes in the System Specifications.
3. Changes in the environment.

4. Errors in the Design frame or skeleton.

5. Changes in the hardware.

6. Typographical errors.

The Maintenance is perhaps the most expensive (costly) Phase of System Development Life Cycle, with respect to other Phases. "The greatest portion, often more than 50 % of the life cycle cost, is spent on maintenance and modification of System to make them do what users had in mind in the first place" [49, p 196]. There is more strong evidence from, "With the majority of the out-of pocket expense to government going into something we call maintenance (60 percent verses 40 percent)" [47,p 21-27]. One reason for these expenses is that Maintenance is concerned with all the Phases of System Development Life Cycle. The Maintenance job is a special case of the Life Cycle, and may have Phases similar to the main Life Cycle starting from Project Proposal for Maintenance, using Analysis & Design tasks, till the last Phase of the Cycle. This work is very similar to that of 'Search for Existing Available Systems'.

The Maintenance generally involves:

1. Addition of certain facilities.

2. Correction of Bugs and operational deficiencies.

3. Making changes in the Requirements Specification which may occur by passage of time. In this case there will be need to start with the Project Proposal similar to

the main Cycle.

4. Making changes in the hardware.

5. Improvement or extension in the System..

The above changes or improvement will involve starting this tasks similarly to the main Cycle. This can be achieved by preparing the Project Proposals and selecting the best feasible one and proceeding as in the main Life Cycle. Any Phase which is not relevent can be bypassed.

The original Design and Implementation Method or Methodology should be used during the Maintenance tasks.

A Maintenance manual should be prepared by the Analyst/Designer together which should describe how carry out the System Maintenance work.

S3 * Prototyping

Prototyping means the construction of a System or a part of a System, with limited goals and selected Requirements, in order to establish some principles, experiment with concepts, determine feasibility or otherwise explore aspects which are not understood.

"Prototyping may be carried out either informally or formally" [68]. A formal Prototype can be developed by using computer and formal System Specifications. An Informal Prototype involves building the Prototype manually [48].

By using a Prototype the User/Management can evaluate and validate some of their Requirements. It can also give to the Analyst/Designer a chance for experimentation and learning with the achievement of specified goals and Requirements in a short period. The successful performance of a Prototype can give confidence to the Analyst/Designer, increasing the chances of successfully ending the project.

Prototyping is a special case of the Life Cycle. It has Phases similar to those of the main Life Cycle. This special Life Cycle can be invoked at any point in the main Cycle, and for a given project many Prototypes may be build. "Prototyping tend to produce a smaller product, with roughly equivalent performance, using less efforts"

[48]. However, this claim is not necessarly true. For example: an operating System prototype for a personal work station could be built from a multiuser mainframe operating System in order to determine which functions or modes of operations were really wanted. Normally the input to the Prototype Cycle is a subset of the currently known Requirements. The further Analysis which may be performed will usually be specific to some aspects of the established Requirements and normally contain small volumes of data. When developing a Prototype these aspects may be examined in depth, without involving Quality measures such as security, accuracy, efficiency, relibility, performance etc..

This Prototype Cycle may continue recursively, bypassing those Phases which are not needed at any particular point. This simple Life Cycle which is often based on trial and error principles usually gives sketchy but informative output. This output can further be improved by using the Cycle recursively especially using the V & V Phase, until the output achieved is acceptable upto reasonable satisfaction. The resultant product then can be used for completing that particular Phase, and continuing development using the main Cycle.

Possibly a suitable testbed and Quality Matrics can be applied to verify a Prototype to improve the Quality. For example a testbed can be applied to access the Requirements, environment, System Specifications, Design etc.. This can possibly be done by evaluating Requirements, Conceptual Design, algorithms, input, output etc.. Quality Matrics can be used to access timing, utilisation of resources, performance, efficency etc..

The following are some specific examples which illustrate where and when to develop a prototype.

1. In all cases a prototype can be developed from a set of available information such as main decisions and constraints. The decisions and constraints which may have global effects should be implemented first in the prototype, before the minor or less important ones.

2. In the Analysis Phase, a Requirements Specification can be derived by developing a prototype of the System which is being developed. Such a prototype can demonstrate to the User/Management how the System will behave and whether is in accordance with their Requirements.

3. An existing available proven good System, if it satisfies the Requirements of the User/Management to a specific extent can be used as a prototype.

4. In some cases developing System Specifications is difficult and time consuming due to lengthy documentation. Such System Specifications become practically impossible to read or amend if needed. This problem can be solved in a better and easier way by developing a prototype of the System, and seeing at an initial level what the System will look like. Redundant and unambiguous System Specifications can also be eliminated by comparing them against the Prototype. This can be achieved by using a V & V Phase with the Prototype.

5. Prototyping can be used in the Implementation Phase. Especially in environments where a small error can cause serious problems such as in defence radar Systems, air traffic control Systems, telephone switching Systems, etc. This situation will be worst if the System is not designed properly. Such a situation can be avoided by using a prototype of the System, to see how it will work in practice.

6. Prototyping can also be very useful in the cases where there is no adequate

information for developing the System. By using a Prototype the User/Management can see how and what is being done, rather than just reviewing the written documents to examine the behaviour of the System.

7. Prototyping can be used to excersise System Specifications and observe their performance. For example to achieve above Tavendale [66] uses a translator program which checks the synthax and symentax of specifications written in FDL, a specification language. He converts these specifications into a Prolog (logic programming language) form. These translated specifications are used in conjunction with a set of modelling functions which are also written in Prolog, to provide an operational interpretation of the sementics of the specification language. These specifications are stored in the Prototyping System, a Prototype of the System may be executed by using a User interface facility which allows the User to excercise the specifications and observe its performance.

S4 * Quality Plan

Quality is a perceived property of an entity and as such is established by consensus. In general Quality is likely to be established relative to a standard.

Quality may be viewed in several ways, sometimes it may be felt rather than measured. It may be considered like beauty which is often in the eye of the beholder. Quality may be established by a number of attributes or factors e.g. readability is dependent on vocabulary, grammer, informal contents and style. It is also possible to have a number of properties which reflect a given Quality. For example complexity may reflect and depend on the readability of text, grammar, information content, functional style, data style, flag setting style, etc.. Some qualities may be transient and therefore may be only meaningful in a particular time band, e.g. reliability.

The conditions that actively contribute to a System having the desired nature, degree of excellence, characteristics, attributes, grade of goodness, inherent features, standard, etc. are called Quality Factors. These factors are often described by adverbs such as reliability, integrity, portability, etc.. However there are some exceptions such as time, money, manpower needed, size of the System etc.. The Quality Factors can actively contribute to the Quality of the System as percieved by the User/Management.

A Quality Plan is a set of procedures, rules, characteristics, constraints which can ensure that the delivered System achieves the required qualities. A Quality Plan may be used to minimise cost of a given System, to control the Quality of each product, to produce output of a consistent standard, etc..

The Quality Plan usually arises as a result of imposing certain specific Requirements by the User/Management. These Requirements may be imposed at the begining of the project or may be notified at later Phases. Such additions to the Requirements occur because the needs of the User/Management may not be constant throughout the entire System development Phases.

The Quality Plan will influence both the production tasks i.e. form of the Life Cycle and the Methods and tools which will be needed. The Quality Plan may influence the Life Cycle by inclusion or exclusion of certain Phases and may decide how rigorously these Phases are to be applied. It may also require specific Methods and tools. For example:

1. a Quality Plan of low level, having low reliability or cheapness, may not require extensive use of the V & V Phase,
2. to develop a very sophisticated System it may be required to do the Analysis Phase very carefully and may hence require the use of SADT,

3. a Quality Plan may require the development of a Prototype first to evaluate some special features,
4. a Quality Plan may require the System to be developed by using MASCOT,
5. a Quality Plan may require the application of a specific Quality control technique before the acceptance of the output of the Phases.

The Quality Plan may be influenced heavily by specific constraints and the System Specifications. For example reliability may be influenced by cost, size and complexity etc.. A sophisticated Quality Plan may give a better System but may involve higher cost. There is a need to have a balance between the required level of Quality and relevant Cost.

A Quality Plan may use forcasting techniques to determine which System development Method/Methodology should be used and which controls should be involved in order to achieve the appropriate levels for the Quality.

In order to demonstrate that the required qualities are being achieved it will be necessary to compare them against predefined events, levels, outputs or standards. The 'quality control' techniques such as reviews, reports, schedules, Sigma charts and Quality Metrics etc. can be used for this comparison.

A System Quality Metric is an indicator or yardstick used for measuring the qualities of a System. The Quality Metrics can measure the Quality in some objective sense and may be in terms of indices. The Quality Metrics in theory can measure quantitatively the degree to which the System posseses a given Quality. The Quality Metrics may be justified for achieving the Analysis and the Design outputs up to a desired level. These indices may be applied to the output of any Phase or subphase in the System Development Life Cycle and to the final product e.g. the System. Their application will assist the Analyst/Designer in checking whether the work is being done according to the required Quality Plan.

Boehm et al [56], McCabe [57] have defined Quality, as a hierarchy of factors and they have tried to measure these factors. There is no standard which can be used universally for measuring the Quality of a System. Different authors have defined how to measure or assess the different Qualities of a System. For example McCabe [57] and Halsted [58] have indicated how to measure the size in terms of complexity. Schach [69] has developed a metric for estimating software size and cost thereof, at the end of Design Phase, of the classical Life Cycle model. Yourdan [20] has used cohesion and coupling the metrics to measure the qualities of modules. Jones [59] and Kitchenham [60] have used error detection and removal statistics as indicators of assessing the certain qualities. "Of particular importance are metrics that measure the efficiency of software life cycle activities and the quality of all representations of software, from requirements specifications to operational run-time code." [p.278, 67]. It has been shown that it is easy to develop metrics which can give misleading results [59]. Therefore development of a System Quality Metric should involve the use of sound statistical and mathematical techniques.

A Quality Plan for a System affects the following:

1. how the Requirements and the System Specifications are to be developed,
2. what Quality Metrics are to be applied during the various Phases of System development and to what extent,
3. the selection of Design Methods or Methodologies and tools to be used during System development.

There may be several Quality Metrics for each Quality Factor and these can be applied:

1. in an existing System to determine the Quality or power of it's attributes e.g. integrity, reliability, etc.,
2. in a proposed System for establishing it's objectives, scope, cost, performance, reliability, etc.,

3. on the output of a Phase, as an indicator to accept or re-do the Phase if necessary. This may involve going back to the appropriate point in the Life Cycle.

The following are some important Quality Factors which may be desirable in a System of high Quality.

1. Maintainability i.e. changeability, amendability,
2. Reliability,
3. Operation (performance, response to an action or event),
4. Portability (applicable to different environment),
5. Robustness,
6. Integrity (completness),
7. Verifiability & Validity,
8. Reusibility,
9. Efficiency,
10. Economic,
11. Simplicity,
12. User Friendly,

etc..

The following are some Quality Factors which can help in ranking one Method or Methodology over others for specific applications:

1. Generality,
2. Applicability (No. of Phases of the System Development Life Cycle covered),
3. Mechanisability (for support and tools),
4. Verifiability & Validity techniques,
5. Tolerability (fault avoidance),
6. Simplicity,
7. Maintainability,

   etc..

It must be clearly stated by the Analyst/Designer what important qualities will be in the System. The User/Management must agree on these before any move. This decision must be based on considerations such as cost, time, manpower, environment etc..

2.3 The Life Cycle In Formal Notation

The following notations are used as:

1. / is used for or,
2. , is used as a separator,
3. The string enclosed between the characters < , > means a task or Phase or subphase,
4. [ ] means is the output of a task, Phase or subphase,
5. ( ) is used for paranthesis,
6. :=: means is defined as,
7. :*:= means is the output of the task, Phase or subphase recursively.
8. < >[a][b] means that the task < > requires as its inputs the out puts of tasks, Phase or subphases <a> and <b>.
9. ([a],[b]) is the collection of outputs from tasks, Phase or subphases <a> and <b>.
10. ([a]/[b]) means the output of tasks, Phase or subphases <a> or <b> is used.
11. The string enclosed between curly brackets i.e. { } means parallel task(s), Phase(s) or subphase(s).
12. If a task, Phase or subphase is not performed then the output of that task, Phase or subphase is defined by the option e.g. if in a case it is not required or desired to do Initial Investigation then this situation will be dealt as:

    [Initial Investigaton]:*:=<P0>[P1]

    where as [P0] and [P1] are the outputs of Null operation and Study of Proposal.

```
<System Development Life Cycle>:=:<SDLC>

<SDLC>:=:<SDLC><Phase Sequence>

         /<Special Case>

         /<Null>

[SDLC]:*:=<SDLC>[Phase Sequence]

          /[Special case]

          /<Null>

<Phase sequence>:=:<Project Proposal Phase option>

                   <Strategy Phase option>

                   <Analysis Phase option>

                   <System Specifications Phase option>

                   <Design Phase option>

                   <Implementation Phase option>

                   <Transition Phase option>

                   <Verification & Validation Phase option>

                   /<Null>
```

Note: The above order enforces the ordering of phases. Each could be null.

```
<Project Proposal Phase option>:=:<Study of Proposal>

                                  <Initial Investigation>

                                  <Initial Feasibility Study>

                                  /<Null>
```

Note: The above sequence orders the operations. Each could be null.

```
<Strategy Phase option>:=:<Strategy Phase>

                          /<Null>

<Analysis Phase option>:=:<Analysis of Environment>

                          <Requirements Analysis>

                          <Prepare Requirements Specification>
                          <Check For Consistency>
                          <Search For Existing Available Systems>
                          <Feasibility Analysis & Detail Proposals>
                          /<Null>
```

Note: The above sequence orders the operations. Each could be null.

```
<System Specifications Phase option>:=:<System Specification Phase>
                                       /<Null>
<Design Phase option>:=:<Planning and Decision>
                        <Conceptual Design>
                        <Physical Design>
                        /<Null>
```

Note: The above sequence orders the operations. Each could be null.

```
<Implementation Phase option>:=:<Implementation Phase>
                                /<Null>

<Transition Phase option>:=:<Transition Phase>
                            /<Null>

<Verification & Validation Phase option>:=:<Verification & Validation Phase>
                                           /<Null>
```

```
<PO>:=:<Null>
[Study of Proposal]:*:=<Study of Proposal>
                       /<PO>
[P1]:=:[Study of Proposal]
```

```
[Initial Investigation]:*:=<Initial Investigation>[P1]
                              /<PO>[P1]
[P2]:=:[Initial Investigation]
[Initial Feasibility Study]:*:=<Initial Feasibility Study>[P2]
                                  /<PO>[P2]
[P3]:=:[Initial Feasibility Study]


<STO>:=:<Null>
[Strategy Phase]:*:=<Strategy Phase>[P3]
                       /<STO>[P3]
[ST1]:=:[Strategy Phase]


<AO>:=:<Null>
[Analysis of Environment]:*:=<Analysis of Environment>
                                /<AO>
[A1]:=:[Analysis of Environment]
[Requirements Analysis]:*:=<Requirements Analysis>([P3],[A1])
                              /<AO>([A1],[P3])
[A2]:=:[Requirements Analysis]
[Prepare Requirements Specification]:*:=<Prepare Requirements Specification>
                                           ([A2],[A1])
                                           /<AO>([A2],[A1])
[A3]:=:[Prepare Requirements Specification]
[Check for Consistency]:*:=<Check for Consistency>([A3],[A2],[A1])
                              /<AO>([A3],[A2],[A1]
[A4]:=:[Check For Consistency]
[Search for Existing Available Systems]:*:=<Search for Existing Available
                                               Systems>([A4],[A3])
                                               /<AO>([A4],[A3])
[A5]:=:[Search for Existing Available Systems]

[Feasibility Analysis & Detail Proposals]:*:=<Feasibility Analysis & Detail
                                               Proposals>([A4],[A3],[A2],[A1])
                                               /<A0>([A4],[A3],[A2],[A1])

[A6]:=:[Feasibility Analysis & Detail Proposals]


<SS0>:=:<Null>

[System Specification Phase]:*:=<System Specification Phase>([A6],[ST1])
                                  /<SS0>([A6],[ST1])

[SS1]:=:[System Specification phase]


<D0>:=:<Null>

[Planning & Decision]:*:=<Planning & Decision>[SS1]
                           /<D0>[SS1]

[D1]:=:[Planning & Decision]

[Conceptual Design]:*:=<Conceptual Design>([D1],[SS1],[A6],[A1])
                         /<D0>([D1],[SS1],[A6],[A1])

[D2]:=:[Conceptual Design]

[Physical Design]:*:=<Physical Design>[D2]
                       /<D0>[D2]

[D3]:=:[Physical Design]


<I0>:=:<Null>

[Implementation Phase]:*:=<Implementation phase]([D3]/[A5])
                            /<I0>([D3]/[A5])

[I1]:=:[Implementation Phase]


<T0>:=:<Null>

[Transition Phase]:*:=<Transition Phase>([I1]/[A5])
```

```
                        /<TO>([I1]/[A5])
[T1]:=:[Transition Phase]


<VO>:=:<Null>
[Verification & Validation Phase]:*=:<Verification & Validation Phase>([P1],
                                        [P3],[A3],[A4],[A5],[A6],[SS1],[D3],
                                        [I1],[T1])
                                        /<Verification & Validation Phase>(P1]/
                                        [P3]/[A3]/[A4]/[A5]/[A6]/[SS1]/[D3]/
                                        [I1]/[T1])
                                        /<VO>([P1],[P3],[A3],[A4],[A5],[A6],
                                        [SS1],[D3],[I1,[T1])
                                        /<VO>([P1]/[P3]/[A3]/[A4]/[A5]/[A6]
                                        /[SS1/[D3]/[I1]/[T1])
[V1]:=:[Verification & Validation Phase]
```

Special Cases

The following are the special cases of the Life Cycle

```
<Special Cases>:=:<Enhancement>
                 /<Maintenance>
                 /<Prototyping>
                 /<Quality Plan>  This is a set of criteria which choses a
                                  Life Cycle
                  /<Null>
```

<EO>:=:<Null>

[Enhancement]:*:=<Enhancement>([I1],[D3],[SS1],[A5],[A4],[A3],[A2],[P3],[PO])

/<EO>([I1],[D3],[SS1],[A5],[A4],[A3],[A2],[P3],[PO])

[E1]:=:[Enhancement Phase]

<MO>:=:<Null>

[Maintenance]:*:=<Maintenance>([I1],[D3],[SS1],[A6],[A5],[A3],[A1])

/<MO>([I1],[D3],[SS1],[A6],[A5],[A3],[A1]

[M1]:=:[Maintenance]

<PO>:=:<Null>

[Prototyping]:*:=<Prototyping><Phases>

[P1]:=:[Prototyping]

Note:

1. Prototyping is building a special case or modifying an existing system. So we need <assessment of the existing system> and the output should feed into a re-evaluation.

b. Also this notation does not cope with parallel life cycles. e.g. <SDLC>:=:<SDLC>{<Prototyping>, <SDLC>}<SDLC> So, <SDLC>:=:<SDLC><Phase Sequence>

/{<SDLC>, <SDLC>}<Phase Sequence>

/<Null>.

<QO>:=:<Null>

[Quality Plan]:*:=<Quality Plan><Phase Sequence>

/<QO>

[Q1]:=:[Quality Plan]

2.4 The Summary Of The Life Cycle

Phases	Starting Point	End Product
1 * Project Proposal	Proposal about what the user & Management want to have.	Recommendations about Analysis Method/Methodology, statement of needs & brief plan, estimates about manpower required, costs, time etc..
*Study of proposal	Study of the Proposal given by the User & Management.	comments, notes, etc., about the proposal and informal recommendations, and a report etc..
*Initial Investigation	Investigating the proposal using the informal report, comments, notes, etc..	statements postulating actual needs, costs, benifits, etc..
*Initial Feasibility study	Output of initial investigation Phase.	Recommendations about Analysis Method/Methodology, statement of needs and brief plan, estimates about completion date, manpower required, costs, etc..
2 * Strategy	output Report of Project Proposal Phase i.e. initial feasibility study sub-phase.	Strategy Plan i.e. how to proceed with the Project.
3 * Analysis	Statements of needs, brief plan etc. obtained from Phase 1	Detailed proposals showing definations of Requirements, Requirements Specification, important parameters, information about environment, etc., statistics about costs, time, manpower etc. and their feasibility to be used for developing System Specifications.

*Analysis of environment	any available information about working, working area, terminologies being used by the User/Management etc..	Short report showing adequate information about the environment working area.
*Requirements Analysis	Report on Analysis of the environment, the statement of needs, brief plan etc..	Statement showing the agreed Requirements of the User/Management.
*Prepare Requirements Specification.	Report of Requirements Analysis	Particular comprehensive and precise statement of Requirements from the User/Management and the environment point of view.
*Ckeck for consistency	Statement of Requirements Specifications	A critically examined report against any contraduction, misleading components, etc..
*Serach for existing available System	The output of sub-phase check for consistency	May lead to Implementation Phase if there is available a suitable System which is according to Requirements Specification.
*Feasibility Analysis and Detailed Proposal(s)	The output of sub-phase check for consistency	Set of detailed proposal(s) giving recommendations, conclusions, statistics about costs, time, etc., and possible benifits.
4 * System Specifications	The selected detailed Proposal i.e. output of the Analysis Phase.	The technical Plan having set of different clear steps to be used as a basis for System Design.
5 * Design	System Specifications which are in terms of set of clear steps expressed in technical terms.	Integrated details about Modules, Sub-systems, their interfaces to other Modules or sub-systems, their controls, constraints etc.. This is called as frame or skleton.

*Planning & Decission	Use of System Specifications.	Basic structure of software and series of hierarichical steps, definations, main functions, processes, operating parameters, I/O data, etc..
*Conceptual Design	Output of subphase Planning & Decission and System Specifications.	Basic assumptions about Design and ideas creating modules, subsystems, data bases, etc., models about entity types, data element types, data structure, data flow diagram, etc..
*Physical Design	Output of subphase Conceptual Design.	Integrated details about data format, data structure, data base, data dictionary, functions, processes, modules, subsystems, their interfaces to other modules, subsystems etc..
6 * Implementation	Frame or skeleton of software System developed during the Design Phase.	Detailed programme statements in suitable programming language(s) and its documentations.
7 * Transition	1. System Specifications, information about functions, real world, etc.. 2. Detailed program statements in a programming language(s) and its documentations.	A working System

2.5 <u>Glossary</u>

<u>Analysis:</u>

The term Analysis is used for the purpose of the systematic study and examination of needs, (obtained in an earlier phase), in order to find, determine, and identify the causes of the problem and Requirements of the User/Management. The results of this Analysis are represented in detailed proposal(s) together with the feasibility report(s) for developing the System Specifications.

<u>Analyst:</u>

An Analyst is a person who identifies the working environment, talks and works with the user and their Management, in order to discover, study and analyse their needs & wants; and prepares proposal(s). The term Analyst may be used as singular and plural and for male and female.

<u>Design:</u>

The term Design is used both as a verb and a noun.

1. As a verb it is used as conceiving & planning by using an agreed proposal(s) and its specifications as input and converting the plan into an unambiguous

executable model, called a System frame or System skeleton to perform its purpose(s) i.e. function(s).

2. As a noun it is used as a tool set which accepts the human proposal(s) and its specifications as input and can develop an unambiguous executable System frame or System skeleton to provide the information to the User/Management and serve his purpose(s) i.e. function(s).

The representation of the Design may be either textual, logical, a diagramatic model or as a Prototype.

Designer:

A Designer is a person who:

1. Uses proposal(s) developed by the Analyst to develop System Specifications,
2. Transforms the System Specifications into a Design to meet the Requirements of the user/Management.
3. Is concerned with designing details of a System, subsystems, modules, etc..

The term Designer may be used as singular and plural and for male and female.

Enhancement:

Enhancement means changing the System due to the development of additional or changed Requirements.

Implementation:

Implementation means the tasks which are concerned with giving practical effect to the Design or System skeleton; the manipulation & translation into code of a software System frame developed earlier in the Design Phase, to fit it to the hardware and also in some cases to the other software. It is also concerned with debugging the code, if necessary. Briefly the Implementation tasks include the following:

1. preparation of an Implementation schedule in consultation with the User/Management,
2. development of accurate & reliable computer programs based on the System frame or System skeleton from the Design Phase.
3. scheduling of modules, subsystems, processes and their mechanisms for invoking and suspension,
4. checking that the required hardware and Software (in some cases) are available and are in working condition,
5. development of operating procedures, manuals, and documentation in a clear and understandable way,
6. training of the end User/Management in the use of the System and the interpretation of the output etc.,
7. helping and guiding the User/Management in developing the organisation and

administrative structure of the data processing centre,

8. ensuring that all the required changes have been implemented and nothing is missing or is ambiguous.

Management:

The term Management is used for:

1. The collective body who handles and conducts the running and operations of an organisation for which the System is to be developed.
2. The authority or controlling body whose Requirements are to be evaluated & understood and the results of the study to be submitted to them.
3. One who directs and handles the System development team.
4. The chief or director of a data processing section or computer section within an organisation.

Maintenance:

Maintenance means the changes incorporated into the Systems at any point from Requirements Specification in the Life Cycle onwards, to rectify faults, bugs, etc. which may appear from time to time. It is also concerned to make the System workable for a particular environment and hardware.

Method:

A Method is a way, set of Phases or steps which should be followed in order to perform certain System development tasks, such as project management, Analysis, Design and Implementation, etc.. It is designed to do certain specific tasks of System development and is available for small or specific types of project. It may be considered as a subset of a Methodology.

Methodology:

Methodology involves philosophies, Management techniques, Methods, Phases or steps, rules, techniques, tools, aids and documentation required to develop a System. In principle it is concerned with the complete span of System development tasks. A Methodology may contain more than one Method or Phases, etc., to do different System development tasks together with the rules to apply them.

Phase:

A phase is a state or stage of development of System tasks over a discrete period of time. Conceptually they do not overlap and preserve time ordering, although in practice this is not usually essential. Each Phase has both input and output.

Programmer:

A programmer is a person who implements the Design decisions in a programming language.

Prototyping:

Prototyping means the construction of a System or a part of a System, with limited goals and selected Requirements, in order to establish some principles, experiment with concepts, determine feasibility or otherwise explore aspects which are not understood.

Quality:

Quality is a perceived property of an entity and as such is established by consensus. In general Quality is likely to be established relative to a standard.

Quality Factors

The Quality Factors are the conditions that actively contribute to a System having the desired nature, degree of excellence, characteristics, attributes, grade of goodness, inherent features, standard, etc.. Quality Factors are often described by adverbs such as reliability, integrity, portability, etc.. However there are some exceptions such as time, money, manpower needed, size of the System etc..

Quality Plan:

A Quality Plan is a set of procedures, rules, characteristics, constraints which can ensure that the delivered System achieves the required qualities.

Quality Metric:

A Quality Metric is an indicator or yardstick used for measuring the qualities of a System. These metrics may be in terms of indices.

Requirement:

A Requirement is something which is deemed to be needed.

Requirements Specification:

A Requirements Specification is a particular and precise statement of those things which are deemed to be needed. This perception of needs is from the view point of the User, their Management and the environment.

Software Tool:

A Software Tool is a program or set of program(s) developed to achieve a specific purpose(s) and task(s) and to support the application of a Method/Methodology.

Strategy:

1. Strategy means a proposed set of actions clearly designed to achieve certain well defined goals over a period of time.
2. In the domain of System development the Strategy means choosing between practices, guidelines, recommended sets of actions and the ordering of development decisions.

System:

1. A System represents a set of interconnected items or elements arranged in a certain order, for example, Management of an organisation, a computer System, a railway network, etc..
2. It is the representation of the Design model or Design frame for an environment.
3. In the domain of computer science a System may be any combination of hardware, software and the User/Management which fulfils the specific function(s) or purpose(s) in an environment.

System Analyst:

See Analyst.

System Specifications:

This is a precise statement of the Requirements in a form which is meaningful to the Designer.

System Specifications Phase:

The System Specification Phase is the Phase which represent an interface between the Analysis and the Design Phases. The Phase which separates completely the Design work from the Analysis and refers to the information which must be delivered from the Analysis to the Design Phase. This Phase converts the detailed proposal into clear and unambiguous statements stating what the System must do.

Transition:

Transition tasks are defined as the evolution and change from one System to another and hence occur only in situations where both exist. Transition tasks are concerned with how the new System should replace the old one and deals with all the problems which may occur during the tasks of switchover from the old to the new System.

<u>User:</u>

A User is:

1. A person who will be using the System such as an operator, software programmer, quality controller, stock control officer, accountant, etc.,
2. A person who will be using the System in a supervisory capacity such as a person who is in charge of a section in a bank, quality control supervisor, etc..
3. A person or organisation who will be using the System as the owner, for example, a government ministry, a banking organisation, etc.,
4. A person who will be using the system without knowing anything about hardware or software, but is supposed to know clearly about the results he is expecting and be able to check the input or output, e.g.,

bank customer, user of flight information services, etc.,

5. The clerical and other people who will work directly with the System by using terminals, filing output forms or interpreting output for their jobs etc.,
6. A person or organisation whose Requirements are to be evaluated and understood and the result of the study is then submitted to them for further actions, e.g., bank authorities, government officials, etc.,
7. Anyone who uses the System by issuing commands.
8. Not the Analyst and Designer and anyone who supplies the System.

The term User may be used as singular and plural and for male and female.

Verification & Validation:

The term Verification applies to the establishment the correctness of something by comparing it against some standard. For example to compare the developed system with its Requirements, System Specifications, etc.

The term Validation is used to ensure that the output of a Phase complies with a standard such as the Requirements, the Detailed Proposals, the System Specifications, etc..

Verification & Validation Phase:

The Phase whose purpose is to:

1. guide the Analyst/Designer and the User/Management in making decisions about their next move, which may be to go ahead or abandon the project,
2. satisfy that the System Specifications are developed according to the result of the Analysis Phase and serve its purpose,
3. satisfy oneself and the User/Management that the System Design is developed according to the Requirements and the System Specifications,
4. check the output of the Design Phase against the basic rules of the selected Method or Methodology, such as rules about input/output data structure, criteria to build modules or subsystems, their connections and calling or exit procedures, etc, and
5. ensure that its result satisfies the test data.

The V & V Phase may be activated at any time and in any Phase of the Life Cycle if needed.

Note:

1. Words like Analyst, Designer, User, Programmer and Management may be used as either singular or plural and male or female. Combinations such as

user/management should be interpreted as a member of either or both groups.

2. The names of Phases, subphases and especially defined terms are used in this thesis with capital letters, except when they are enclosed in quotation marks.
3. Any other term used and not defined specifically, is used in its common meaning in the software engineering.

CHAPTER THREE

3 Summary of the Results of the Analysis & Comparison

This Thesis is an attempt to identify the problems which Software Engineers are facing when using existing Methods/Methodologies. In the past where different groups have studied the Methods/Methodologies each member of the group has studied one or two Methods/Methodologies taking into consideration certain specific goals & objectives, commercial interests and personal beliefs. In order to obtain a unique and un-biased approach we developed a set of questions without having any commercial and personal interest. Since the whole study was carried out by the same individual consistency and uniformity of the study was achieved thus eliminating commercial & personal bias towards or against any Method/Methodology. The selection criteria of these Methods/Methodologies was on the basis of their popularity in commercial, industrial and academic environments. Maximum possible effort was made to collect the working manuals, published materials and research papers on these Methods/Methodologies. Letters were written to the concerned people, personal contacts were made and library facilities were used to collect the maximum possible information about the existing Methods/Methodologies. Unfortunately very little information could be obtained on some Methods/Methodologies and was not sufficient to cover and do the feature Analysis on certain other Methods/Methodologies. This constraint plus lack of time & resources forced us to reduce the number of elements covered in this study. Considerations were made to find out those aspects which are most unique and original in the Methods/Methodologies. It was found that in general there is no Method/Methodology which can be used for developing a System for all different types of problems and environments and is capable of satisfying or likely to satisfy different User's Requirements and cover all the tasks of System Development. The importance of having a global framework was recognised which could be adopted in all different environments and problems and perform all different tasks of System Development having by-passing facilities, if some Phase is not essential for a particular problem.

The absence of any mathematical foundation, quantitative concepts, unified tasks done by the System Development Methods/Methodologies forced us to do comparative feature Analysis of the complex and varying nature of the Methods/Methodologies. The Global Life Cycle framework was used as one of the basis of our study. The framework which hopefully can be adopted successfully by different Methods/Methodologies and in different environments.

One of the most difficult problems faced in the study was due to a lack of a standard unified & comprehensive set of terminologies. It was very difficult to find out how a specific concept is termed and defined in one Method/Methodology and how it relates to a similar concept in the other Method/Methodology with usually different terms. We tried to study these terms, their definitions and concepts in which these are being used. Efforts were made to solve to a limited extent the problem of these terminologies & their definitions. Later it was felt necessary to have a global set of terminologies and their definitions and use them as a basis of comparision. We developed and used a set of certain important and commonly used terminologies & their definitions, which are of a global nature. This was supplemented with the information about the contents & concepts of the terminologies in the Methods/Methodologies. This set of standardised and global terminologies and their definitions if enlarged could be helpful in future while developing new Methods/Methodologies for different environments. This needs further research work and extension. This could be done with the support of some international body.

The use of a common framework and set of questions resulted in identification of similar features, common terms, concepts, specific areas of applicability and their commercial claims and its validity. The success of this approach made it clear that this model framework and set of questions can also be applied in other Methods/Methodologies which have not been covered in this study.

It was found that the most of the

Methods/Methodologies do not mention specifically, explicitly and distinctly their objectives, scope and fundamental principles. We made a concerted effort to determine these objectives, scope and principles.

Information about the pre-requisites, starting points, and final product of the Methods/Methodologies were not given explicitly and clearly and also these were found to be not the same. In the study we determined that JSD starts the tasks of System Development by developing the model of the real world. This model is developed on the basis of complete information about the real world including the required Functions. Therefore, before starting the System Development tasks it is assumed that such information is available by the User/Management. The final product of JSD are the Specifications for the programs. LSDM generally starts the System Development tasks from the Feasibility Study report. But in certain cases if the Feasibility Study report is not available it may be developed by LSDM. The final product of LSDM is the Physical Design which is the detail about files/DBMS definitions, program Specifications, operating schedules, etc.. MASCOT starts its tasks by developing an overall Software Design on the basis of operational Requirements. The pre-requsit for this task is the availability of host and target computers. The host computer must have filing, text editing, compiling and diagnostic facilities. The final product of MASCOT is a complete implemented Software System on the target computer. ETHICS starts System

Development tasks by diagnosing the human & technical needs of the User/Management of an existing System which may or may not be computerised. Therefore, an old System must be there to start the task of System Development. The working and evaluated socio-technical System is the final product of ETHICS. SADT starts its tasks from a most general description of the System contained in a Module, represented by a box. To develop a detailed System; statement of System Requirements, decisions, constraints and information about Functions are essentially pre-requsite. SADT finally delivers a SADT System Design model. This diagrammatic model contains a set of activity and data diagrams which show the activities and data aspects of the System and identifies the components of the Software System. SA starts by investigating the existing System which may or may not be computerised. To do this some proposals from the User/Management showing their wants are pre-requsites. The final product of SA is the Structured Specifications which is also called Functional Specifications. SD starts its tasks from the output of SA i.e. a statement showing what the System is supposed to do, this is also called Structured or Functional Specifications. The end product of SD is the final Structure chart and data Dictionary.

It was found that JSD, LSDM, ETHICS, SADT and SD offer clear information about the steps which are followed during the System Development activities. However, it was found that the MASCOT Method is one which just provides brief directions showing how to develop a System, whereas SA does not offer any step by step clear & precise directions.

The study revealed that the phases of the Life Cycle and the steps/phases of the Methods/Methodologies in general do not have one to one correspondence. In most of the cases the Methods/Methodologies do not cover the whole span of the Life Cycle and also not a single one covered the complete tasks of System Development. In such cases the Methodologies are considered as a subset and a particular case of the Life Cycle Model. For example JSD & MASCOT completly ignore the tasks concerned with the Project Proposal, Analysis and Transition Phases. These Methods/Methodologies are more or less concerned with the System Specifications, Design and Implementation tasks. LSDM covers certain aspects of Project Proposal, Analysis, System Specifications & Implementation, and gives more emphasis to the Design tasks. There is no evidence that the tasks of the Transition Phase are covered by LSDM. The ETHICS Method partialy covers the Phases of the Life Cycle except the System Specifications Phase which is not covered at all. In fact ETHICS completly ignores the technical aspects of the System Development and gives more emphasis to the Analysis aspects of System Development. SADT ignores the tasks concerned with the Project Proposal, Implementation and Transition Phases of the Life Cycle. It was observed that SADT is mainly concerned with Development of System Specifications and Design tasks.

SASD covers the Analysis, System Specifications, Design and Implementation tasks of System Development. SASD also does not cover the Project Proposal and Transition Phases. It was found that most of the Methods/Methodologies have failed to find and define the problems of the User/Management, hence obviously the solution obtained by them are mostly not satisfactory and do not satisfy the User/Management. They also failed to do the tasks concerning the Project Proposal. For example they do not consider time, schedule, financial constraints, etc. to avoid possible delay or financial problems. Some Methods/Methodologies confuse the Analysis tasks with the Design and do Analysis tasks superficially. This situation leads to misleading results, making the User/Management unhappy and causing frustation. Finally not a single Method/Methodology in the sample has covered the Strategy Phase of the Life Cycle which is in fact a new concept given in the Life Cycle.

The study revealed that JSD, LSDM, ETHICS, SADT and SD have fairly clear guide lines which can help during the performance of the System Development tasks according to their scope and coverage. It was observed that LSDM has a number of rules and guidelines which are relatively better than other Methods/Methodologies. MASCOT Methodology does not show the guidelines and details. Similarly SA has certain limited general information showing how to proceed with the Analysis tasks.

While collecting the information about the special training required to use the Methods/Methodologies it was found that JSD, LSDM, MASCOT, SADT emphasise the need to have special training before using their Method/Methodology. ETHICS and SASD do not have any such conditions.

JSD, LSDM, MASCOT, ETHICS, SADT and SASD all have certain tools/supports to be used at various levels during the System Development. The tools/support used by JSD, LSDM, MASCOT, SADT were found not to be an integral part and are optional, whereas the tools/support used by ETHICS and SASD are integral parts. In SASD tools/support can be selected according to the size & type of a problem etc.. How effective and useful their tools are is difficult to say. Perhaps it may need another study.

JSD, LSDM, MASCOT, ETHICS, SADT, SA do not specify any rule which can help in choosing a suitable Method or approach according to a particular environment. The main reason for this is that these Methods/Methodologies have limited scope and can deal with certain class of problems and are problem oriented. They follow a similar approach in all cases. SD is a little exception because it offers two different options to be used in the selection of tools/support i.e. transform analysis & transaction analysis. The Methods/Methodologies under study were found not to be flexible enough to fullfil the wider scope of demands and environments. It was difficult to determine whether a given Method/Methodology can be applied perfectly for a particular environment. It was not possible to evaluate the performance of these Methods/Methodologies and to say 'this is the best' Method/Methodology suitable for all purposes. Some might be better than others in certain specific environments and work excellently but may fail in other cases.

JSD is found to be suitable for developing a System to solve data processing, Process control, real time and other sequential problems. LSDM and SADT claim to be suitable for a wide class of problems. MASCOT is found to be better for developing large real time Systems. ETHICS and SASD are mainly suitable for commercial and industrial problems.

JSD, LSDM, MASCOT, ETHICS, SADT and SASD were found to be machine and language independent. However, it was noticed that MASCOT which was developed with particular reference to CORAL-66 can be used only by the languages which have facilities similar to CORAL-66.

JSD, MASCOT, ETHICS and SASD were found not assisting in the Project Management and Strategy Phases. Authors of LSDM claim that their Methododology assists in Project Management and have a technique to assist in this direction but they do not develop or use any Strategy for System Development. SADT assists in project Management to a limited extent.

It was found that JSD has its own approach which is quite different from the traditional approaches, and uses the structured concept differently. LSDM, MASCOT and SA follow the Top-down approach. SADT and SD use the Top-down Modular concept. The ETHICS Method does not use any such traditional approaches.

While studying how and to what extent the Methods/Methodologies apply the V & V activities or something similar to it, we found that JSD uses the V & V activities but not precisely and significantly. It develops the System and validates it by asking the User/Management whether the real world is like the model they have developed. LSDM applies a limited concept of V & V activities. LSDM achieves this by checking and proving the System Logic. This is achieved by identifying the event that may change the System data, validating the Requirements for each event etc.. MASCOT makes some effort to maintain quality assurance. This is done by developing test strategies during the Design tasks and using the concept of structured decomposition. ETHICS uses V & V activities by evaluting the performance of the System against the set of objectives fixed at the begining of the project. It also verifies the efficency of the System after its Implementation. The efficency is tested by checking that there is improvement in the control of variance and that no other variance has been created during the System Development tasks. SADT and SASD do not use the V & V concept or any thing similar to it. It was felt that in general these Methods/Methodologies can not guarantee to deliver reliable & error prone Systems. The reason for this is that these Methods/Methodologies generally do not have the V & V Phase and proper testing facilities as an integral part.

It was felt that JSD is suitable for small to medium size problems. LSDM, ETHICS & SASD are suitable for varying sizes of problem. MASCOT is claimed to be suitable for large real time Systems. SADT is also suitable for large and complex problems, but it can also be used for smaller problems.

While collecting information about the communication facilities which the Methods/Methodologies employ it was found that JSD, LSDM, SASD do not have any specific & particular communication facilities. In LSDM the Analyst/Designer and the User/Management can consult each other at various levels, if needed. The ETHICS Method communicates with the Analyst/Designer and User/Management by documenting the results of the Analysis & Design tasks. MASCOT and SADT use documentation facilities for this purpose. SADT also has well defined communication facilities, but these facilities are limited to the use of the Analyst/Design.

We examined the documentation facilities, i.e.

tools, symbols, notations and their style, readability, unambiguity, effectiveness, etc.. It was found that JSD uses a set of diagrams, notations and symbols for the purpose of documentation. These diagrams are a by-product of various System Development tasks. JSD notations & symbols are easy to draw, read and in general are clear. JSD rely on symbols which are supported by a special JSD textual representation style, which makes documentation unambiguous. But JSD documentation is not suitable for the use of the User/Management. LSDM uses a set of forms for the purpose of documentation. It is claimed by the authors that the LSDM documentation is a by-product of the System Development tasks. There is no evidence to support the success and effectiveness of these symbols & notations. LSDM does not rely too much on symbolic formalism. It has symbols and notations similar to JSD & SASD. MASCOT has very specific and precise documentation rules & standards which reflect its approach. It has a set of diagrams and six different symbols & notations. The style of these symbols & notation can help effectively in communicating during the System Development tasks. The symbols are not designed for the use of the User/Management. MASCOT relies too much on its symbols & notations and therefore pays more attention to developing its diagrams. The ETHICS Method does not have any specific documentation tools or facilities. It uses some general symbols and notations for the purpose of documentation. The style of the symbols and notations are similar to the flow chart. ETHICS does not rely too much on symbols & notations. SADT uses the graphical language called **'SA'**. This language gives details concerning Analysis & Design gradually and unambiguously. The language uses forty separate notations and conventions for the purpose of documention. The language has a well defined and systematic set of rules, symbols & notations which are easy to draw and read. SADT **'SA'** language and its rules, symbols etc.. have a limitation in that these are not suitable for the use of the User/Management. SADT highly depends and relies on the use of these symbols. SASD has good & well defined documentation facilities but these facilities are not suitable for the use of the User/Management. The notations used are clear, readable and easy to understand, but they are not perfect. SASD does not rely on symbolic formalism. The Methodology also uses textual representation for documentation. In general we did not find a single Method/Methodology which has an impressive documentation technique starting from documenting the User's Requirements to the end product. Their documentation causes problems when Enhancement and Maintenance tasks are done. It is suggested that in future while developing the new Methodologies efforts must be made to improve communication and documentation facilities to allow comprehensive and meaningful communication and documentation.

The answer to the question about the experience needed to use the Method/Methodology showed that the Designer using JSD must have some sound experience in order to express Process structure and Process communication. The Designer must be capable of understanding the environment and communicating with the User/Management. LSDM has the limitation that it cannot be used without proper training, therefore, its authors provide training facilities. To use the MASCOT Methodology the Designer must be experienced and qualified, otherwise there is a risk that the resultant System may not be optimal and may high overheads. The ETHICS Method does not require any special experience and training only general experience and knowledge is needed. The Analyst/Designer using SADT must have a clear concept of the Methodology which can be achieved by studying the literature and completing excercises. SADT is in fact easy to understand and follow. To use SA the Analyst must be experienced. The level of experience required varies according to the size of the problem to be solved and its environment. In general SA is easy to use. For SD the Designer must be highly experienced. There is a great risk that an inexperienced Designer in complex problems may lead towards inefficient Systems.

It was observed that the Enhancement and Maintenance tasks in JSD can be done in certain cases only i.e. when the addition or replacement of a Function is required in the model of the real world. But if it is required to make changes in the existing Function Process or change the model, it becomes more difficult to do so. LSDM covers Enhancement by taking the logical view of the current System and extending or replacing it with a System which includes further facilities required by the User/Management. The Enhancement and Maintenance tasks are possible in MASCOT and they are performed similarly to those of the initial integration of the System. It can change, create and introduce new sub-systems without any major disturbances. Maintenance in the ETHICS Method can be performed by repeating all the steps of the Method. The Enhancement and Maintenance tasks in SADT can be performed by identifying in the diagrams the activities and data structures which are to be introduced, changed or modified. This involves the whole Analysis/Design steps of SADT. Because it involves the re-design of data structure etc.. SASD can do Enhancement and Maintenance tasks by adding or changing a Module if needed. But it may create many ripple effects. In general the Methods/Methodologies were found not to have efficient and impressive Enhancement & Maintenance facilities.

Investigation of the prototyping facilities available in the Methods/Methodologies showed that JSD uses a concept similar to prototyping by completing model specifications and adding Functions to the model. It does not use any tool/support for this purpose. MASCOT also uses the prototyping concept; it also does not have any tool/support. LSDM, ETHICS, SADT and SASD do not use Prototyping during the System Development tasks.

About the use of the concept of Quality Plan and

Quality Metrics it was found that JSD, ETHICS and SADT do not use any Quality Plan and Quality Metrics. LSDM does some regular checking procedures etc. without applying any Quality Metrics. MASCOT uses the concept of Quality Plan without applying any Quality Metrics. SA does not use the idea of Quality Plan or Quality Metric, but SD uses indirectly the concept of Quality Plan and Quality Metric by applying the criteria of modularity, and using coupling and cohesion as measures. These two measures are qualitative not quantitative.

It was found that JSD, MASCOT, SADT and SASD do not involve the User/Management during System Development. LSDM and ETHICS involve the User/Management, but at varying levels. ETHICS is mainly based on active particaption of the User/Management during the System Development tasks. LSDM supports to some extent the involvement of the User/Management during the System Development tasks but not actively like the ETHICS Method.

During the search for the unique approach which these Methods/Methodologies follow it was found that JSD firstly uses an approach by which a model of the real world is developed with which the System must be concerned. This model is developed without considering the Functions at the beginning. The Functions are introduced into the model at a later stage. LSDM considers three different views in parallel i.e. data flow, data structure and events during the System Development tasks. No other

Method/Methodology has adopted a similar approach. Further LSDM is based on a data driven approach rather than on Function. MASCOT uniquely develops Software first on a host computer and then implements it on the target computer. Such Software can be implemented on a single Processor or set of inter-connected Processors. Further it is possible to use a subset of MASCOT facilities out of six facilities. In ETHICS the active Participation of the User/Management, during the System Development tasks, having democratic philosophy in designing a System, considering the human & technical aspects of the System equally and the idea of using variance Analysis are the unique approaches which no one has used earlier. SADT uniquely supports and does the tasks of System Development by team work. It involves the reader-author iteratively during the Development of diagrams for the purpose of communication and review. The unique concepts used by SASD are coupling and cohesion.

In examining whether the Method/Methodology is a combination or variant of others it was found that JSD is an extension of JSP. LSDM evolved from the LBMS data base Design tool and SSADM which is a mandatory Methodology for Government Departments in U.K.. MASCOT is in evaluation. MASCOT-3 is a varient of MASCOT-1 and MASCOT-2. ETHICS and SADT are not a combination or variant of any other Method/Methodology. SASD is a combination of SA and SD.

CHAPTER FOUR

4.1 Summary

In the last few years there have been a number of different efforts to analyse, evaluate and determine the strong & weak points of System Development Methodologies. These efforts adopted three different approaches. The first two attempts were made separately by IFIP WG 8.1 [63] and the Dept. of industry, UK., et. al. [78]. These studies were made by examining and comparing the suitability and competence of Methodologies against their solution for a particular problem. For this purpose IFIP WG. 8.1 developed a model problem about an hypothetical conference. The authors of the Methodologies had to present a paper showing how they could solve the model problem. The second group i.e. the department of industry, UK. et. al. decided to develop two different problems to study how Methodologies can solve them. These problems were about the development of an aircraft monitoring System and a KAPSE database System. This approach had many drawbacks and could not give unbiased results. For example the idea of developing a problem which could equally be used by all these Methodologies (which were developed for specific purposes and having different scopes), was not a suitable approach. This is because practically it is impossible to develop an unbiased model problem, no matter how carefully it is formulated. Further the span of the System Development Life cycle which could be used to solve the particular problems was limited and not complete. For example, Requirements Analysis was excluded because the Requirements were already presented in the problem statement. This limited the coverage of the System development tasks. Many Methods i.e. not Methodologies were also included in this study. Therefore, the efforts to analyse and evaluate the facilities, strong and weak points, suitability and competence of these Methodologies were not successful.

The second approach was to develop a list of model questions and apply it on selected Methodologies to identify and study their characteristics and features. These selected characteristics and features were used to compare and evaluate the individual Methodologies. This approach was adopted first during 1981 jointly by the department of industry, UK. and five other UK. based organisations [78]. These organisations applied this approach when they wanted to study how the Ada language can best be used by System Development Methodologies and how successfuly these Methodologies can solve problems related to the Ada environment. This study was further assisted by a review committee comprising of eleven other members each representing his organisation. They developed a set of 13 characteristics against which 21 Methodologies or their variants were examined. Later during 1982 Wasserman and his group conducted a mail survey [82], [83] on slightly different lines but for a similar purpose. Due to commercial reasons it is difficult to assess how honestly the authors of the Methodologies have answered the questionaire. Of course no one would like to express the weak points of his Methodology and will admit that his Methodology is not suitable for the Ada language. Further the framework of these two studies was itself subjective and biased towards the suitability of Ada; it is therefore difficult to say how reliable and unbiased the results of these studies were.

During 1983 IFIP WG. 8.1 organised a second conference to do feature Analysis of System Development Methodologies [64]. The WG. at this times adopted the second approach mentioned above. At this conference the feature Analysis of some of the Methodologies from the first conference, plus some other Methodologies was made. In this conference the solutions of the model problem of the IFIP first conference [63] were compared and analysed by various participants. This comparison and Analysis of the Methodologies was performed against a number of features and on the basis of the information about the goals, applicability, origin, experience, etc. of the Methodologies. The main drawback in this study was that different participants tried to compare and analyse the features of the Methodologies without having a common frame, framework or list of common features. It was therefore difficult to relate and compare their work with each other and find agreed common points of interests. Further what constitutes a Methodology was also not clear in all the above studies.

The third approach was to carry out a feature Analysis of the Methodologies by using a set of questions uniformly on each selected Methodology, without doing comparative analysis, and express these features in uniform terminologies. This approach was adopted by Maddision and his group during 1983 [65]. This study was done collectively by eight people. Each one wrote at least two internal papers on the basis of which their joint work was published. They selected nine Methodologies to study their features and essentials. Three of these are covered in this thesis. Maddision and his group gave more coverage to the Analysis activities, reflecting the orientation of the Methodologies. The group did not mention the criteria for the selection of the Methodologies. Their definition of Methodologies was very wide and is not related to a complete span of System development tasks. Some Methods were considered as Methodologies e.g. JSD which is defined as a Method by its authors. Other important aspects such as principles, Maintenance, Enhancement, Quality Plans, Quality Factors, Quality Metrics, Verification & Validation activities, level of participation of the User/Management, etc. were not covered. Little effort was made to cover prototyping. The terminologies used were also not clear and suitable for the different Methodologies.

Each of the above studies were done by group of people having different experience, ability, understanding and personal liking. Also the possibility of them having commercial interests cannot be ignored. It is therefore, difficult to comment on how reliable, effective and consistent these studies were within particular scopes and objectives. Furthermore all these studies did not cover the complete span of System development tasks. In some of these studies either the Analysis aspects were ignored, concentrating on Design and Implementation tasks, or concentrated more on Analysis tasks giving less emphasis to Specification, Design and Implementation tasks. Not a single study covered the complete span of System development.

In order to do the Analysis and Comparison of these Methodologies for the purpose of this thesis it was felt to be essential to adopt a better approach; an approach which is free from strategic omission and errors. In order to carry out the study properly it was felt to be desirable to:

a. cover the important and commonly used Methods and Methodologies.
b. cover the complete span of System development i.e. from initial conception of needs to final realisation of System, and to use the Phases of a System Development Life Cycle as a standard for comparison i.e. to know what Phases of a life cycle the Method/Methodology covers.
c. have a set of terminologies with clear

definitions and to apply these consistently, uniformly, properly and uniquely,

d. apply a comprehensive questionaire on the Methods/Methodologies covering all the essential characteristics & features to do a feature Analysis and Comparison of the results,

e. perform an Analysis and Comparison of Methods/Methodologies consistently, free from any commercial or personal interest, and free from strong personal believes in support of or against any Method/Methodology.

At the begining of this study several Methods/Methodologies were selected for the Analysis and Comparison. The criteria for selection was their popularity in commercial, industrial and academic environments. Comprehensive and utmost efforts were made to collect their working manuals, published materials and research papers. Correspondence and personal contacts were made with the concerned persons. The library facilities were used to collect maximum possible information about the Methods/Methodologies. Whenever possible, seminars and lectures given by the authors of these Methods/Methodologies were attended. Unfortunately, often very little information could be obtained. The reasons for this were either these Methods/Methodologies did not have enough materials to be supplied or they were not interested in being exposed against their claims. This constraint and lack of time forced the study to a limited number of Methods/Methodologies. Six Methods/Methodologies were finaly selected for the study.

To adopt the above new approach it was necessary to select a System Development Life Cycle which can cover the complete span of the development tasks. Several life cycle models were examined which were used by various Methods/Methodologies as an integral part such as SASD [5,18], SADT [11], NIAM [14], LSDM [16], HOS [31], There were also some life cycle models defined explicitly such as those defined by Enger [36], Biggs [37], Blum [38], Ramamoorthy [39], O'Neill [40], Freeman [41], etc.. It was found that these life cycle models have several serious problems, e.g. not covering the whole span of System development tasks. They were found to be suitable only for certain types of problems and many have lengthy Phases which may increase the probablity of committing errors. They had no by-pass options (i.e. skipping unwanted Phases). The lack of a suitable and complete life cycle model forced us to develop a new System Development Life Cycle model which is of a global nature and can be applied appropriately on all class of problems and to different environments. The model which is developed in this thesis covers all these requirements and is free from the above problems. This model is not restricted to any specific System Development Method/Methodology. The model consists of a set of Phases which are generalised ones and can help the Analyst and Designer to do their tasks. This model starts from the Project Proposal and covers all aspects of System development such as Project Proposal, Strategy, Analysis, System Specification, Design, Implementation, Transition and Verification & validation. These Phases are clear and flexible, any Phase can be by-passed or modified, if needed, according to the environment and type of problem. Iteration and recursion are two main properties of this model. The model also has the capability to deal with special cases such as Enhancement, Maintenance, Prototyping and the inclusion of a Quality Plan. To do this the model has four special cases which can be applied at any time, if needed. These special cases are particular instances of the life cycle. This life cycle model is based on the active participation of the User/Management. The model offers guide lines and gives some idea how to start the development of a System and to Implement it. Any Phase can be by-passed if found unnecessary. The model specifies clearly the inputs/outputs of each Phase. The life cycle model is also expressed in a modifed form of BNF notation. All these features made the life cycle model unique and of a global nature but at the same time a standardised one.

The frequent use of the same terms for different purposes and different terms for the same concept by the Methods/Methodologies made it difficult to do the Analysis and Comparison. To resolve this problem it was at first decided to use the IEEE standard glossary of software engineering terminologies [34] as a hallmark or a basis for the Analysis and Comparison. Later it was observed that the IEEE glossary itself is not complete & comprehensive and cannot serve the purpose. Efforts were made to find out the discrepancies in the terminologies, such as how different terminologies for the same thing are being used, and how similar names are used for different purposes. The terms which these Methods/Methodologies use very often and in various ways with out having any definition were identified. It was then decided to develop a glossary of important terms with clear definitions and apply them in this study. Therefore a set of 29 different terms was developed to express certain concepts in a unified manner so that these can be used globally for the purpose of comparison. These terms are also used in the life cycle model. This enabled the Analysis and Comparison of the Methods/Methodologies in a consistent and uniform manner, free from any discrepancy.

To do the Analysis and Comparison of the selected Methods/Methodologies which are of varying scope, nature and having different features, it was decided to list all the important features and characteristics which should be analysed and compared. Later it was found that it is not feasible to cover all the features or characterisics. The reasons for this were the varying nature of the Methods/Methodologies, time etc.. Finally a set of 35 selected questions was applied. The last two are concerned with definitions and terminologies.

The use of the above approach resulted successfully in the identification of objectives, scope, concepts, specific area of applicability, etc. of the Methods/Methodologies. It also helped in determining how true are their commercial claims.

It was observed that most of the Methods/Methodologies did not mention explicitly and distinctly their scope, fundamental principles, etc.. Information about the pre-requisites, starting points and final outputs were not the same for different Methods/Methodologies

Most of the Methods/Methodologies which were studied provide information about the steps which they follow during the System development tasks. Only MASCOT and SA were found not to give step by step, clear and precise directions.

The study revealed that the Phases of the life cycle and step/phases of the Methods/Methodologies in general do not have a one to one correspondence. In most cases they do not cover the whole span of life cycle. Not a single Method/Methodology covered the complete tasks of System development. No Method/Methodology has used the strategy Phase.

All the Methods/Methodologies expect MASCOT generally provide clear guide lines which help during the performance of System development tasks according to their scope and coverage.

It was observed that all these Methods/Methodologies emphasise the need to have special training before using them. However, ETHICS and SASD were exceptions.

All the Methods/Methodologies have certain tools/support to be usedat various levels of System development. These tools were not necessarily an integral part of the Methods/Methodologies.

It was found these Methods/Methdologies in general do not specify any rules which can help in choosing an approach suitable to a particular environment. The reason for this was found to be that these Methods/Methodologies have limited scope and can only deal with certain class of problems and are hence problem oriented. However, SD is an exception, since it offers two different options i.e. transform and transaction Analysis. The Methods/Methodologies were found to be not sufficiently flexible to fulfil the wider scope of demands and environments. It was not possible to evaluate the performance of these Methods/Methdologies and to say 'this is the best one', i.e. one which is suitable for all purposes. Some of these might be better than others for certain class of problems and environments and work ideally but may fail in other situations.

JSD was found suitable for data processing, process control, real time and other sequential problems. LSDM and SADT may be used for wide classes of problems. MASCOT was found to be better for developing real time Systems. ETHICS and SASD were found to be suitable for commercial and industrial problems. This conclusion appears consistent with the claims of the respective authors.

These Methods/Methodologies were found to be machine and language independent. However, MASCOT is an exception and is best with languages which have facilities similar to the CORAL-66.

All the Methods/Methodologies except LSDM and SADT do not assist in the project Management and Strategy Phase. The authors of LSDM claim that they do assist in project Management and have appropriate techniques. SADT assists in the project Management to a limited extent.

JSD was found to have its own unique approach which is quite different from the other traditional approaches such as Top-down, Bottom-up, etc.. Other Methods/Methodologies follow traditional Top-down approaches. The ETHICS Method does not use any such approach.

JSD, LSDM, MASCOT and ETHICS all apply the concept of V & V activities or something similar to it but not precisely and significantly. SADT and SASD were found not to be applying a V & V concept or anything similar to it.

It was felt that JSD is suitable for small to medium size problems. LSDM ETHICS, SADT and SASD were found suitable for varying sizes of problems. MASCOT was found suitable for large size real time Systems.

JSD, LSDM, SASD were found not to have any specific and particular communication facilities between the User/Management. ETHICS, MASCOT and SADT use their documentation facilities for this purpose.

JSD has notable documentation facilities but these are not easy or good enough for the User/Management to be able to understand them. LSDM uses a set of forms for documentation purpose. The symbols and notations which LSDM uses are similar to those of JSD and SASD. MASCOT has very specific and precise documentation rules and standards and therefore, relies too much on its symbols and notations. ETHICS does not have any notable and influential documentation facilities and therefore does not rely too much on its symbols and notations. SADT has detailed and well defined graphical language called 'SA'. SASD provides good and well defined documentation facilities but these facilities are not easy for the

User/Management to understand. In general no Method/Methodology was found to have impressive documentation facilities and were not good enough when Enhancement and Maintenance tasks were to be performed.

It was felt that to use JSD, MASCOT and SD is not easy. To use other Methods/Methodologies is not difficult but they may need some training and experience.

It was found that the Enhancement and Maintenance tasks can be dealt with to some extent by these Methods/Methodologies. But they do not provide any efficient, notable and impressive Enhancement and Maintenance facilities.

No Method/Methodology was found using the Prototyping facilities during System development except JSD and MASCOT but these two also do not have any specific tool/support for this purpose.

No Method/Methodology was found using the concept of a Quality Plan or Quality Metrics. However, LSDM and MASCOT apply some vague concept of Quality. SD applies the concept of coupling and cohesion during System development tasks this may help to some extent in improving the Quality of a System.

No Method/Methodology except LSDM and ETHICS involve the User/Management during System development tasks. ETHICS is mainly based on active participation of the User/Management during System development.

The JSD approach of developing a model of the real world was found to be unique. LSDM considers three different views i.e. data flow, data structure and events in parallel during System development. No Method/Methodology has used all three views together. The development of Software on a host computer and Implementing it on the target computer is a unique tactic in MASCOT. The active participation of the User/Management is a unique approach in ETHICS. SADT uniquely supports and does the tasks of System development by team work. The concept of coupling and cohesion made SASD unique.

While examining whether the Methods/Methodologies are a combination or varient of others, it was observed that JSD is an extension of JSP, LSDM evolved from LBMS database Design tool and SSADM. MASCOT is in its third variation. SASD is the combination of SA and SD. ETHICS and SADT were found to be generic ones.

4.2 Suggestions

Almost all sciences have an agreed standard set of terminologies and corresponding definitions. The study of System development Methods/Methodologies revealed that software engineering is a science which does not have a complete, comprehensive and standard set of terminologies and corresponding definitions. These Methods/Methodologies use different terms for the same concept and the same word for different concepts. These Methods/Methodologies also use certain terms without defining them explicitly. The IEEE standard glossary of software engineering terminology also does not cover a large number of the terms used by these Methods/Methodologies. The terms which are defined by the IEEE often differ from the way these are used by the Methods/Methodologies. This results in communication problems especially when two or more Methods/Methodologies are used in the same organisation. This also causes problem in teaching, training and research environments. Due to commercial and prestige reasons it is probably impossible for the existing Methods/Methodologies to ammend their terminologies.

There may be two feasible approaches which can be adopted to solve the problems of terminologies. The first one, which is short term and a temporary solution is to develop a set of terminologies and their definitions in such a way that these terms can represent a wide range of concepts and possibly be suitable for all different types of Methods/Methodologies. This seems to present difficulties and can only be developed for certain selected terminologies i.e. only for those terms which have the same word but are used for different purposes. We have made an effort in this direction and a glossary of terms has been developed for certain important commonly used words. Another but long term feasible solution for this problem can be to develop a detailed and complete set of terminologies and corresponding definations. This can be done by getting technical and financial co-operation from international non-commercial scientific and research organisations such as the IEEE Computer society, the BCS etc. and with the help of academic and experienced experts. If it is possible to have a unique standard set of terminologies and their definitions in other fields of science and engineering than why can it not be possible in software engineering. These terminologies later if possible, should be legally backed through the EEC, UNO, UNESCO and other inter governmental organisations, etc.. The Methods/Methodologies should be checked by the authorities to see how close they are to the standard. Any lack of standard should be recorded, reported and communicated to consumer councils pointing out clearly how, where and to what extent there is a deviation from the standard. The international organisations should be promoted by giving them financial support to develop new multi-purpose Methodologies based on an agreed standard set of terminologies using the concept of a global System

Development Life Cycle such as the one developed in this thesis.

Software engineering is in its initial stages and developing fast. The scientific, commercial and industrial factors are forcing the software engineers to develop new Methodologies. These Methodologies are essentially required to be free from the existing deficiencies and limitations and in future they will replace the existing Methodologies. Before conceiving new Methodologies it is essential to provide for the software engineers a complete, comprehensive and standard glossary of terminologies.

The System developers must accept that the present existing System development Methods/Methodologies apart from their general deficiencies are incomplete. They can only develop Systems partially i.e. covering either technical or human aspects of a problem. In fact very few Methods/Methodologies are available which consider the human and Management aspects of a problem and encourage the participation of the User/Management during System Development i.e. the group who will be using and will mainly be concerned with the System. Further, to do the job of System development effectively it is essential that the Analyst/Designer must be competent in both aspects of the System i.e. technical and human areas. Unfortunately during our study of the Methods/Methodologies it was revealed that the Analyst/Designer were not expected to specifically know about both areas. Therefore, in this study of System development Methods/Methodologies we included the ETHICS Method which covers the human aspects of the problem while developing a System. But the study of the ETHICS Method revealed that it gives too much importance to the involvement of the User/Management during the development of a System and very little to the technical aspects. The main drawback in this Method is that it lacks the ability to deal with technical aspects of the System development tasks. For example, the ETHICS Method has a great disadvantage in that it cannot cope with determining the technical Requirements and developing Requirements Specification and other vital technical aspects of System Design. With such an unbalanced involvement there can arise certain technical problems which may result in a conflict. Especially in the case where the User/Management does not have enough technical information and want to impose their constraints, conditions and needs etc. for prestigious reasons.

There is an acute need to have a new approach which can avoid the human ill-effects, fullfil the human needs of the User/Management and cover all the important and useful technical aspects of the traditional System development Methods/Methodologies as its integral part. A Methodology is required which can cover the technical as well as human Requirements equally. There is a need to have a marriage between such a Method and a technical

Method/Methodology which misses the Management and human aspects of a problem and does not cover the Analysis activities in its scope. For example the JSD and ETHICS Methods can be integrated to compliment each other. In the absence of such an approach there is a great risk that the growing computer industry in future will reject such Systems which do not cover both aspects of System development.

It is suggested that there is a need to develop an automatic Software Tool which can be used as an integrating tool for using different System development Methods/Methodologies according to the global System Development Life Cycle Model. Such a tool will enable everyone to get benefits from different Methods/Methodologies at different levels of the Global Life Cycle. To integrate the Methods/Methodologies it will need to have a language. Any suitable high level language can be used for this purpose. Such a tool can be built first by developing a prototype and then the package itself. The available proven good tools which support the selected Methods/Methodologies can also be used and integrated in the selected Methods/Methodologies. Such an integrated tool will hopefully help in solving the System development problems, e.g. Structured Design, ETHICS & JSD, etc..

APPENDIX A

A.1 Introduction

This appendix Analyses and Compares the essential features of the selected System development Methods/Methodologies. The selection of these Methods/Methodologies is based upon their popularity in the computer world. The purpose of this Analysis and Comparison is to determine the objectives, scope, characteristics, features, etc. of these Methods/Methodologies. A set of 33 selected questions is used for this purpose. These questions determine, analyse and compare the objectives, scope, characteristics, features, etc. of these Methods/Methodologies. These questions also examine, determine and evaluate the extent to which these Methods/Methodologies cover the Phases of the Global System Development Life Cycle i.e. cover the span of System development tasks. The order in which these Methods/Methodologies are presented in this thesis is alphabetical, not according to their popularity or any other factor.

A.2 ETHICS Method

The following general questions concerned with a Method/Methodology are applied to evaluate the strong and weak points and to perform a feature Analysis of the ETHICS Method [9], [76], [77], [87].

1. What are the objectives and scope of the Method?

Objectives:

The following are the objectives of ETHICS Method:

a. To provide an approach to do systematic Analysis of the problem required to be solved.

b. To provide the means to identify and meet human needs of the User when designing a Computer System.

c. To provide an approach which can give a solution to get or improve both job satisfaction and efficency needed by the User/Management.

d. To have a System Development approach which can either eliminate work problems or ensure that they could be quickly corrected.

e. To provide socio-technical approach of System development which can show how technical and human Requirements can be analysed and System Designed, which people both can and will be pleased to use.

Scope:

The ETHICS Method according to its definition is concerned with the development of an Effective Technical and Human Implementation of Computer System. In brief it covers the following:

a. Identification of human & technical needs and helping the Analyst/Designer & the User/Management in setting the human and technical objectives.

b. Identification of the variances which can affect and influence the performance of the existing System.

c. helping and assistng the Analyst/Designer and the User/Management in how to Design technical and human objectives.

d. Design a new System according to the objectives and implement it.

e. Monitor and evaluate the performance of the new System and examine whether it is implemented according to the objectives.

Note: ETHICS Method does not show in sufficient detail how to identify the technical needs and develop the technical objectives i.e. Requirements Specification. The Method is not concerned with the development of System Specifications and technical aspects of the System Design. The Method emphasises more on the Analysis aspects of the existing System.

2. What are the fundamental principles of the Method?

The following are the fundamental principles adopted by ETHICS Method.

a. Provide opportunities to the User/Management in redesigning their own System by active participation.

b. The role of Analyst/Designer should be to help the User/Management in the development of a System. The User/Management should do the main task of the System development.

c. The User/Management responsible for System development should represent all grades of the staff in the organisation.

d To achieve an agreed solution involve, if possible, participation of the trade unions, if there are any.

e. A 'unit operation' should not be split. A group of User should take responsibility for one or more 'unit operations' in their section.

f. Ensure that corrections are made as close to the source of variance as possible.

Note: The development of the technical aspects of a System is generally not possible by the User/Management due to lack of professional knowledge and experience. The above principles can successfully be applied mainly during the Project Proposal, Strategy, Analysis and some aspect of Implementation & Transition Phases.

3. What are the pre-requisites for the Method?

The old System must be there to start the Analysis work and obtain diagnostic data.

Note: The ETHICS Method has limited scope in the sense that it does not show what to do when designing a new System.

4. What is the starting point of the Method?

ETHICS Method starts the System development by diagnosing the human & technical needs of the User/Management of an old existing System, which is not working smoothly, efficiently and satisfactorily. The old existing System may or may not be computerised.

Note: In their literature and the reports of case studies which were made by the authors it is assumed that technical Requirements are readily available.

5. What is the final point or step of the Method, and what is the deliverable of the Method?

The System evaluation is the final step of the

ETHICS Method. The Method delivers a working socio-technical solution in place of the old System.

6. What are the steps or phases of the Method?

ETHICS Method has ten steps which are essentially taken during the Project Proposal, Analysis and some Implementation & Transition tasks. The following are the steps of the ETHICS Method, and its brief description.

a. Diagnosis.

b. Socio-technical System Design Objectives.

c. Setting out alternative solutions.

d. Setting out possible Socio-technical solutions.

e. Ranking Socio-technical solutions.

f. Preparing a detailed work Design.

g. Accept best possible socio-technical solution.

i. Implementation.

j. System Monitoring.

k. System Evaluation.

In this Method in principle equal efforts are made to identify technical and social needs and constraints.

In the first step efforts are made to diagnose those 'needs' which can effect the long term Job-satifaction needs. To achieve this information the 'needs' are collected by conducting a survey. A questionaire is developed and used to find about the job-satisfaction. The questionaire is answered by the User. The information obtained from the questionaire is examined and analysed analytically by using a tool i.e. the 'Job-satisfaction Analysis Model' [8], [76], [77] which has several different measures. These measures are information about 'fit' between what the User are seeking from the organisation and what they will achieve if their wants are made. The 'fits' which are generally measured are about knowledge, psychological, efficency, task-structure and ethics. If this 'fit' is found to be good in a pre-change situation, it means that job satisfaction is high and the chances of having conflict between the User and Management will be minimal. Otherwise efforts are made to diagnose why the fit is not good. This is done by analysing the outcome of the result of the questionaire. The reasons for the lack of fit as well as the social and technical problems are determined, which the new System must avoid. Social & technical Requirements are determined which should be in the new System. A democratic approach is adopted by making small groups of the participants i.e. the User and the Management, analyse and discuss the outcome of the questionaire.

In the second step the diagnostic data on job satisfaction and technical Requirements are used as the input for the second step. This data is used for stating in detail the human and technical needs, the area where job satisfaction can be improved and technical factors which can improve the System performance. The

Analyst/Designer identifies technical and social constraints and resources available to use them to specify two sets of technical and social objectives. At this level these objectives can state in detail policy and guidelines for the development of a Design and also a check against which the System can be evaluated. The compatibility between the two sets of objectives is then checked. Careful consideration is made in consultation with the User/Management on the impact of the outcome on the affected departments or sections.

In the third step the two different sets of possible solutions are prepared independently in accordance with the social & technical objectives. Each set gives alternate solutions in terms of technical/social aspects of the problem. The advantages and dis-advantages of each alternate solution is determined. Each solution is evaluated separately and independently against the objectives established in the second step. The feasibility of their Implementation is also determined. From these two sets a set of feasible social and technical solutions is prepared to be used as a short list.

In the fourth step each solution listed in the short list of alternate feasible social and technical solutions are compared with each other to determine which 'fits' are compatable i.e. technical solutions are compared with social solutions. The pairs which are in harmony are selected and listed in a set of possible socio-technical solutions.

In the fifth step the socio-technical solutions are examined and ranked by the User/Management to determine the best solution. The amount of effort to be spent on this task depends on time, money and the type of the problem. Cost-benefit Analysis of different solutions is made in order to help in the selection of the best socio-technical solution. The selected socio-technical solution must be evaluated by the User/Management against the job satisfaction needs established in step 1. Further the advantages and dis-advantages must also be considered in relation to the social & technical objectives. The selected solution must 'fit' together and achieve both human & technical objectives i.e. it must jointly meet in the best possible way the technical and human objectives.

In the sixth step structure of the tasks to be performed by the User is considered. Efforts are made to assign the duties in a way that work group or individuals have a logically integrated pattern of work activities. To achieve such a pattern the ETHICS Method recommends some job-design principles such as the principles of task variety, skill variety, task identity, task autonomy. The Method also emphasises to consider the following point at the time of assigning the tasks:

a. the User must know about their job and how it should be performed,

b. the social environment must be developed in a way that the User feel socially happy and feel that their work is being recognised,

c. the User must not feel that his promotion or future will be blocked.

The job structure which can provide to the User an acceptable level of job satisfaction should be chosen as the accepted solution of the job structure. Such structure can only be achieved by the active involvement of the User/Management and Analyst/Designer.

After selecting the best possible socio-technical solution and developing the job structure according to the given guidelines efforts are made in the seventh step to re-examine the needs. Any thing which must essentially be in the System and found missing should be re-considered to see if it can be satisfied in some way. This is done by skipping through all the previous steps.

The Implementation is the eighth step of ETHICS Method and starts when the System Design work has been completed. This step involves a set of operations which change the existing System smoothly and successfully from one organisational structure to another. Implementation is achieved by forming a single group from the User/Management and the Analyst/Designer under the leadership of one of the most experienced section leaders in the department. (It does not rely entirely on the Analyst/Designer.) The group represents the section and consists of a number of multi-skilled clerks and specialists. The new organisational structure and technical aspect of the Design must essentially be Implemented in a way that they fit well with the expectation of the User/Management. The Implementation approach of the Method requires creation of an attitude which regards the new System as being better than the old one, and the User's recognition of the competence of this new System. All concerned must understand the nature of logic of the change consequences. The Implementation step provides to the User/Management a commitment to the new System explaining how it works and giving training and skill to operate it efficiently. The step also provides the job-training, which is generally done by the most experienced staff taking part in the System development group.

In the System monitoring step tasks of System Implementation are monitored i.e. what is happening and how it is happening. In this step careful efforts are made to ensure that the System which is being Implemented is valid and staying in accordance with the human objectives i.e. human aspects of the Requirements Specification set in earlier steps. If there is a descrepancy between what is to be intended and what is being done then controls are made to bring it back in the right direction.

In the System evaluation step it is observed closely what has happened, to ensure that the Design has been implemented according to the objectives set earlier. This step is essentially applied to avoid and rectify mistakes, if there are any, and study the human aspects of System Implementation for future purposes. The diagnosis tool [8], [76], [77] i.e. 'Job-satisfaction Analysis Model' is used once again to evaluate the level of fit between the User's needs and the Management's job Requirements in the new System. If the resultant value of the 'fit' is better than the change, it means that the new System can lead towards better job satisfaction, so it can be accepted. Otherwise there is need to take action which may be to redesign the System to the desired level of 'fit' i.e. to control and change back to the right course of action.

Note:

a. From where the technical Requirements are obtained is not clear. In their literature and the reports about case studies which were conducted in the past by the authors, it is mentioned that technical information was provided by the Management. Perhaps a reason for this was that the participants of the case studies did not have the technical knowledge to find the technical needs and the set of technical objectives. Further it seems that the ETHICS Method has emphasis on designing the social aspects of a System,

b. the System objectives are used as the basis of establishing development policies and a standard against which the final System can be evaluated,

c. the Method has provision for System Maintainance which is done iteratively.

d. This Method is in its evolutionary process and lacks many aspects of System development. For example it does not show

how to develop the technical aspects of a System, though in principle the Method agrees to do so.

e. The Method looks time consuming and therefore expensive.

7. Are these steps or Phases of the Method systematically procedurised, or do they just give broad directions?

Yes, these steps are Systematically procedurised and must be followed during System development.

8. What Phases of the System Development Life Cycle does the Method cover?

ETHICS Method covers partly the tasks concerning the Project Proposal, Analysis, Design, Implementation and Evaluation Phases of the Life Cycle.

The tasks concerning with the Project Planning Phase of the Life Cycle are covered partially by the ETHICS Method and overlap with its steps. Tasks concerned under the heading of The Study of the Proposals are not performed. The Method covers the Initial Investigation subphase 1.2 by diagnosing why there is no job satisfaction and what the User want. To achieve this the existing System is investigated during the 'diagnosis' step of the Method. This is done by conducting a survey to find out the most frequent problems which are in the existing System and cause job dissatisfaction. The reasons for dissatisfaction are traced out, the User's wants are reviewed and social & technical problems are determined. From the above information the efforts are made to find the User's actual needs which should be fulfilled in the new System. Technical Requirements and radical changes are determined which should be in the new System. In the 2nd step of the Method details of these human & technical Requirements are stated by reviewing wants and seeking out actual needs. From the technical and human Requirements the technical and social objectives are determined. These objectives are checked for their correctness & completeness. The impact of the outcome of these objectives on different departments are carefully considered in consultation with the User/Management. In the 3rd step of the Method two different sets of possible proposals are prepared from the objectives each showing social/technical solutions.

The Initial Feasible Study subphase 1.3 of the Life Cycle is performed partly in the 2nd step of the Method. This is achieved when the Analyst/Designer identifies social/technical constraints, resources available and determines the feasibility of these objectives.

The Method does not incorporate the activities concerned with the Strategy Phase of the Life Cycle.

The tasks concerned with the Analysis Phase of the Life Cycle are done partially. The subphase 3.1 i.e. the Analysis of Environment is done by involving actively the User/Management. Efforts are made to collect & understand the information about the environment where the existing System is working and to know about the application area for which the new System is being designed. The factors which affect the environment and are responsible for creating incorrect data, performance etc. are also determined. All these are done in the 'diagnosis' step.

The tasks concerned with the subphase 3.2 of the Life Cycle i.e. the Requirement Analysis are done by identifying clearly whether or not the identified needs will actually be required and detailed information about the social and technical Requirements are collected. The security, social, privacy, managerial and financial aspects of the needs are also determined and communicated to all concerned in the first two steps of ETHICS Method.

The tasks concerned with preparation of the Requirements Specification i.e. the subphase 3.3 of the Life Cycle are done by ETHICS Method during its 2nd step. This is achieved by setting detailed technical and social objectives, policies and guidelines for the development of a Design. These objectives define what the System must do to satisfy the human and social Requirements and also can be used as a check against which the final System can be evaluated. These objectives are used in transforming the Requirements into System Design. This is done by setting out alternate social/technical Design solutions. Each solution acts as an initial model which can exhibit as much as possible the necessary information for forming a

Design.

The tasks concerned with the subphase 3.4 i.e. Check for Consistency is done during the 7th step of the ETHICS Method. This is achieved by making efforts to re-examine the User's needs, Requirements Specification, job description etc.. Anything which must be there and is missing is re-considered to see if it can be satisfied in some other way.

The tasks concerned with the subphase 3.5 i.e. the Search for Existing Available Systems are not covered by the ETHICS Method.

The tasks concerned with the subphase 3.6 i.e. Feasibility Analysis and Detailed Proposals are done first at the time of developing the objectives during the 2nd step of ETHICS Method. This is done by ensuring that these objectives are realistic and capable of achievement without any major difficulty and verifying that resources are available for implementing them. Second time the task of Feasibility Analysis is done during the 3rd step by setting out the alternative social and technical solutions. At this point efforts are made to take into considerations the resources which are available and constraints which must be met. The activities concerning the development of a set of various proposals which are feasible to achieve the Requirements Specification called objectives by the Method, are done during the 3rd, 4th and 5th steps of the Method. This is done by developing two different sets of possible proposals in accordance with the social and technical objectives i.e. Requirements

Specification. The advantages & disadvantages and feasibility of Implementation of these two sets of proposals is determined. From these two sets a set of alternate feasible social and technical solutions is prepared. These alternate feasible social and technical solutions are compared with each other to select a set of the best socio-technical solutions. In the 5th step these socio-technical solutions are examined and ranked by the User/Management to determine the best solution.

The tasks of the System Specifications Phase of the Life Cycle which is concerned with stating clearly, un-ambiguously in the technical terms of the computing environment, what a System must do, are not covered by ETHICS Method.

The tasks concerned the Design Phase of the Life Cycle are to conceive & plan out, by using an agreed proposal and its specifications as input, converting the plan into an un-ambigous executable model, called System, to perform its purpose(s) i.e. function(s). These above tasks are not performed by the ETHICS Method, as it does not develop a System Specification in technical terms. This is an important area which is missing in ETHICS Method.

The tasks concerned with the Implementation Phase of the Life Cycle which are the manipulation, translation & debugging of code, for a software System or skeleton developed earlier in the Design Phase, to fit to the hardware and in some cases software also, are mainly not covered by the Method. The Method does not follow any

Implementation Strategy or approach such as JSP, Top-down, Bottom-up, etc.. The Method covers the training, helping and guiding the User/Management in the use of the System and interpretation of the outputs. Guiding the User/Management in the organisation and administration of the data processing center. Ensuring that all the required changes have been implemented in a way that they fit well with the expectation of the User/Management and nothing is missing or is ambiguous. These tasks are done during the 8th step of the Method.

Note: ETHICS Method has a great weak point that it does not show how to re-write the selected proposal in the technical terms and make the technical foundation of the Design. The System Specifications Phase which acts as interface between Analysis and Design tasks are missing. There is no way to know how to get technical input for the Design tasks. This can create many Implementation, Transition and Maintenance problems. Unless the Method is supplemented by some other Methodology, for developing the technical input of the Design i.e. System Specifications, the System will not be reliable and many Implementation problems may arise.

9. Does the Method give equal weight to the different Phases of the System Development Life Cycle?

No, ETHICS Method gives more weight to the Analysis Phase. Very little weight is given to the Design, Implementation & Transition Phases of the Life Cycle. The tasks concerned with the Strategy, System Specifications are not covered by the Method.

10. Does the Method contain a set of guidelines which can lead from the start of the project to the Implementation & Transition of the System?

Yes, ETHICS Method gives guidelines on how to proceed and do the work of System development within its scope. The Method is silent about how to get the System Specifications and other technical aspects of a System.

11. Does the Method require any special training before using it?

No, ETHICS Method does not require any special training. However, it expects from the Analyst/Designer that they have a good knowledge of different ways in which work can be organised in a System. They must know about System Management techniques.

12. What Methods, tools and support are used by the Method?

The Method and tool which are used by ETHICS Method are the Variance Analysis and the Job-satisfaction Analysis respectively. The following is a short description about these:

Variance Analysis:

It is a simple Method [8], [77] for identifying technical weaknesses in an existing System i.e. for identifying weak areas of a System. This tool gives priority to identify and eliminate the 'key variances' first i.e. those variances which are most direct or important in their impact, cost, quantity and Quality. The success of a new computer System is determined by the level of extent to which prechange variances have been eliminated or better controlled without creating new variances.

Note: The variance Analysis is not concerned with the temporary problems such as machine break-down, strike, shortage of man power etc..

Job-satisfaction Analysis:

This tool [8], [76], [77] is used to analyse the variables which are concerned with the human aspects of the problem such as knowledge, psychological, efficency, job-structure and ethical factors. In short efforts are made to determine:

a. What the individual seeks from his employer, if his most important needs are to be meet.

b. Up to what level an individual wants to persue his own interests and is unwilling to compromise.

c. The extent to which an individual wishes to behave in a unique and individual way i.e. to show his individuality.

d. The level of desire of an individual to recognise his personal qualities, promotions, etc..

e. How flexible the working hours are according to individual's Requirements to satisfy himself.

The fit between employees needs, expectations and job experience is determined for the above five variables. If the fit between the User's need and expectations are met then it means that the User have high job-satisfaction. If the fit is not entirely good then efforts are made to further analyse why this is. In the new System considerations are made to avoid the reasons of lack of fit. This is achieved either by amending personnal policies as well as job-design. The knowledge and job-structure fit can be improved by creating forms of work organisation and job-structure in the light of a socio-technical approach of System Design.

13. Does the Method offer clear and precise rules for choosing the Methods to be used in a particular environment?

No.

14. For what class of problem is the Method most suited?

The Method is suitable for commercial and Industrial problems.

15. Is the Method language and machine dependent?

No.

16. Does the Method assist with the project Management and Strategy?

No.

17. Is the Method based on principles which are in general:

a. Structured,

b. Top-down,

c. Bottom-up,

d. Others?

No, ETHICS Method does not use any of the above concepts.

18. Does the Method use the V & V activities or something similar to them?

Yes, and it is performed by using the set of objectives specified in its earlier steps as a standard for evaluating the System.

19. To what extent deos the Method use the V & V activities?

ETHICS Method verifies the efficiency of the System after its Implementation. The efficiency is tested by checking that the improved control of variance has taken place and that no other variance had been created.

20. Is the Method suitable for small/medium/large Systems?

ETHICS Method can be used for developing the problems of all different sizes.

21. Does the Method have communication facilities between the User/Management and the Analyst/Designer?

Yes, ETHICS Method communicates with the Analyst/Designer and User/Management by documenting the results of the Analysis/Design. These documents diagramatically propose changes in the nature or arrangement of hardware, work-flow, group social structure of the User/Management, individual job structure etc..

22. What documentation facilities, tools does the Method have?

The Method does not have any specific documentation tool or facility.

23. Does the Method use any symbols or notations, for

expressing the outputs of the Phases, e.g. specifications, Design, etc.? If yes, what are they?

Not any special one.

24. Is the style of symbols or notations readable, can these symbols or notations lead to effective, rapid and unambiguous communications?

Yes the style of symbols is readable and is similar to flow chart. The set of symbols or notations is effective.

25. Does the Method rely on highly symbolic formalisms?

No.

26. What experience is required by the Analyst/Designer in applying and using the Method?

Not any specific ones, only general experience is needed.

Note: From the reports of the case studies, it was found that ETHICS Method is not much concerned about the development of the technical aspects of the System Design such as System Specifications, etc.. Perhaps no special technical experience is needed by the ETHICS Method.

27. Does the Method have provision for covering special

cases such as:.

a.	Enhancement?	No,
b.	Maintenance?	Yes,
b.	Prototyping?	No,
c.	Quality Plan?	No,
d.	Search for Existing Available System?	No.

The Maintainance activities can be performed during the sixth step of ETHICS Method. In this step any thing which the System does not provide to meet the technical and human needs of the User/Management and were derived from the Analysis are re-examined by repeating all the previous steps.

28. What Prototyping tools/support does the Method have?

The ETHICS Method does not use the concept of Prototyping.

29. Does the Method incorporate a Quality Plan for a System?

No.

30. Does the Method use any Quality Metrics?

No. But there is a tool which measures the 'fit' analytically for different variables such as knowledge, efficency etc..

31. Does the Method support the User/Management participation? If yes, to what extent?

ETHICS Method is completely based on active participation of the User/Management. It involves not only the User/Management but even their trade union, if there is one, during the System development. The Method constrains the active participation of the User/Management during the System development. This participation is not only up to a voting level but it is much more than that. It involves thinking, knowledge and competence. It is concerned with working jointly.

32. Has the Method some unique approaches? If yes, what are they?

Yes, the following are the unique approaches adopted by ETHICS Method during the System development.

a. the active participation of the User/Management and trade union and giving emphasis on the need for meeting the human needs of the User when developing a computer System,

b. the emphasis on the approach to have democratic philosophy in which staff of the concerned department are asked to

Design their own work organisation with the help of the Analyst/Designer,

c. ETHICS Method takes both technical and human factors into account during the System development,

d. ETHICS Method has a Variance Analysis facility which no one has used before.

Note: It does not offer a technical solution of the problem.

33. Is the Method a combination or variant of other Methods? If yes, which ones?

No, ETHICS Method is not a combination or variant of other Methods.

A.3 JSD Method

The following general questions concerned with a Method/Methodology are applied to evaluate the strong and weak points and to perform a feature Analysis of the JSD Method [21], [22].

1. What are the objectives and scope of the Method?

Objectives:

To develop a System which can represent & reflect the real world with which the System is concerned. This System should not depend on the functions or the concept of functionality.

Scope:

JSD is concerned with the problems which are of sequential nature, and have time ordering. The problems which have no time ordering are out of its scope, e.g. census etc.. JSD excludes activities such as Project Proposal, Strategy, Systems Analysis, etc..

2. What are the fundamental principles of the Method?

   a. A formal description of the real world with which the System is concerned must be developed before specifying the functions.
   b. An adequate model of a time-ordered real world must itself be time ordered.
   c. The System should be Implemented by

transformation of formal specifications into runable System having efficient & convenient set of processes adaptable to running on the available hardware and software.

d. A dynamic real world cannot be modelled accurately and sufficently by a database, which is an appropriate medium for the static models.

3. What are the pre-requisites for the Method?

The complete Information about the real world including the required functions should be available from the User/Management.

Note: JSD does not mention how to get the complete information about the real world. They claim that "the model implicitly defines a coherent set of possible system functions" [p.8, 21]. But the problem is that they have not defined the term function and have not mentioned how to find and represent a function. In fact JSD is not tackling this problem differently as they claim. JSD also ignores completly how to determine the various parts of a problem and their relationship with each other.

4. What is the starting point of the Method?

The real world is regarded as a fixed starting point which is based on general and informal information provided by the User/Management.

Note: The information provided by the User/Management to develop a System is usually not complete, adequate and some times incorrect. A System which is based on such information may not be reliable and in some cases may not work correctly.

5. What is the final point or step of the Method, and what is the deliverable of the Method?

Implementation is the final step of JSD. It creates the specifications for the programs which can be developed further by using JSP.

6. What are the steps or phases of the Method?

JSD divides the activities of System development into two major steps i.e. developing specifications and developing Implementation.

JSD develops the specification for the System, by building it up from parts which are themselves sequential processes. The activity of development of System Specifications involves two major steps i.e. specify model and specify functions. These two steps at their lowest level are further divided to develop System Specifications. These sub steps are as follows:

a. Entity/Action step,

b. Entity structure step,

c. Initial Model step,

d. Function step,

e. System timing step.

In the first two steps an abstract decription of the real world is developed by describing the relevent aspects and eliminating the irrelevent ones. The subject matter is defined by describing the real world in terms of entities and the actions which they perform or suffer taking into consideration the ordering of these actions. The abstract description of the real world is achieved by producing lists of entity types and action types and a short description about them. Further a diagramatic statement of the ordering of these actions for each entity is also needed for this purpose. These diagrams are called structure diagrams. The result of these two sub steps is called an abstract description of real world.

In the initial model step the abstract description of the real world in terms of sequential processes, is realized as a process model. "A sequential process is no more than a statement of time ordering of events" [p.1, 21]. The Designer begins to specify the System itself by specifying a simulation of real world. This is done by having for each sequential process in the real world a corresponding sequential process in the System. The output of the initial model step is a System Specification Diagram. This diagram shows a model of processes, connected by the real world entities which they model.

In the Function step the known functions are expressed in the form of processes each one in isolation from other functions, with long life time. These function processes are added to the System. If a function is simple it can easily be added to the model process by fixing the operations directly into the process text, which may be needed to get output from the function. Difficult and complicated functions may require new processes to be added to the System. This process can be connected to the model process by using process connections which may either be data stream connection or state vector connection. These function processes receive inputs either from model processes or sometimes, from extraneous data streams. Sometimes a function may require more than one function processes. What function process are needed, and how they should be connected in SSD is decided in this Function step. In this step the structure of each function is also decided. The purpose of function process is to Provide System outputs.

In the System timing step the issues about timings are considered. These issues are concerned with the function outputs. A JSD model lags behind the real world because the messages reaching from the real world take time to reach the System boundary. This time lag may be different at the different parts of the System. The Designer in this step settles with the consent of the User/Management the timing Requirements and specifies what delays are acceptable for the various parts of the System. The result of this step is used at the

Implementation step

The output of the major step i.e. development of the specifications, achieved after performing the above five steps is written in a set of sequential processes. These processes specify a model of the real world and functions thereof. The System Specification Diagram depicts the processes, their connections with each other and input/output at System boundary. The detailed specifications of each type of process is also written in structure text.

The Implementation step, which is the second major step of JSD, does not have any sub steps. This step is concerned with transforming the specifications written in terms of a large set of sequential processes to make it executable on available hardware and software, if there is any. These sequential processes can be implemented on any available machine irrespective of the available number of processors in it. This can be achieved by sharing the processors, scheduling, inversion and internal buffering. To do these a special purpose scheduling program is developed. The Designer determines the number of real or virtual processors which may be needed to execute the System. The Designer decides how the System processes are to be allocated, shared in time and scheduled on the available processors. Different scheduling decisions may be made according to the type of problem and the environment. These decisions may give different Implementations for the same JSD specifications. During JSD implementation activities the Designer develops a

System Implementation Diagram. This diagram depicts the outlines how the System Specifications are transfered to do Implementation. This diagram is concerned with the technical purpose and not for the use of the User/Management. The results of the JSD implementation step may be conventional. It can give the output in terms of set of programs, there may be a data base of master records and batch and on line programs to update them etc..

Note: In practical situations the abstract description of the real world is generally developed on the basis of the information provided by the User/Management and the technical preconception about what a System will look like. These descriptions are developed without getting detailed information about the problem and the activities which come under the System Analysis. Generally the User/Management do not know much about their actual problems, needs and its solution. It is, therefore, obvious that this approach can work on small/medium problems, and will not be suitable for big problems.
Different Designers may develop different abstract descriptions of the same real world. Because the criteria to elimimate the irrelevant aspects of the real world depends entirely on the choice of the Designer. This may lead towards drastic errors in the absence of detailed System Analysis activities. Further the situation may get worse if the Designer is new for the particular environment, which is very common.
What is function? How to determine and represent it? is

not explained clearly by JSD. Perhaps they represent a function by expressing it in an imperative English statement.
The Implementation activities in JSD are problem and environment oriented. Further the other important issues concerned with the Implementation are not dealt with by JSD implementation. Perhaps because their specifications are not written in terms of other issues such as in terms of on line transaction modules or batch, and on line programs to update them.

7. Are these steps or Phases of the Method systematically procedurised, or do they just give broad directions?

The JSD steps are systematically procedurised. But the definition and coverage of the steps differs from that of the Phases of the System Development Life Cycle. The activities of these steps are overlapping with respect to the Phases of the Life Cycle.

8. What Phases of the System Development Life Cycle does the Method cover?

JSD does not cover all the Phases of the System Development Life Cycle. It only covers the activities of the System Specifications, Design and the Implementation Phases to some extent. The coverage and activities of these Phases are mixed together and many Design decisions are taken in its 'implementation' step.

9. Does the Method give equal weight to the different Phases of the System Development Life Cycle?

No. The Method covers only the activities concerned with the System Specifications, Design and Implementation Phases of the Life Cycle. It gives more weight to the activities which comes in System Specifications and Design Phase. The activities concerned with the Implementation Phase are partially covered. For example the following activities are covered during the development of the specifications:

1. The information given by the User/Management are converted into an abstract description of the real world.
2. During the development of specifications the terminology and the pragmatism of the application area are converted into the terminology and the pragmatism of computing. This is achieved during the development of processes etc..
3. A JSD specification step can express in precise and consistent form what the System will do on the basis of the information provided by the User/Management.
4. A JSD model of a real world developed during the specification step can determine the System boundaries and its interface.

5. JSD specification specifies details about processes and operations of the System. These are expressed by SSD and structured text in English.

Most of the System Specification activities expressed in the Life Cycle are covered by the JSD specification step.

Note: There is no evidence to show that error handling and its recovery can be made by the specifications developed by using JSD.

Most of the Design activities expressed in the Life Cycle can easily be made by JSD. For example:

1. The activities concerned with the Planning & Decision sub Phase can be done by JSD during the development of specification and Implementation steps, e.g. description of the real world in terms of sequential processes and their time ordering, deciding which functions should be added to model etc.,
2. The activities concerned with the sub Phase Conceptual Design can be done in JSD by developing processes network, establishing how information will flow into the model, expressing the details about the state vectors and the data streams etc.,
3. Some activities concerned with the sub Phase the Physical Design can be done by

developing details about processes, adding the functions into the model or developing and adding the function processes, etc.

Note: JSD does not use the concept of decomposing the problem into sub problems to get subsystems which may be necessary if the problem is complicated and is of very large size. Further JSD does not have any explicit testing provision which can be utilised.

JSD implementation step partially covers the Implementation activities as expressed in the Life Cycle. For example:

1 It realizes the abstraction of the real world and manipulates in more concrete terms by using inversion and internal buffering techniques,

2. JSD transforms the Design frame or JSD model for implementing according to the available hardware and the software, if there is any,

3. JSD implementation step schedules the sequential processes according to the available number of real or virtual processors,

4. JSD implementation step allocates the processors to the processes according to the available time resources,

The JSD implementation step does not cover many other Implementation activities as expressed in the Life Cycle. For example:

1. preparation of an Implementation schedule in consultation with the User/Management,
2. checking that the required hardware and software (in some cases) are available and are in working condition,
3. development of operating procedures, manuals, and documentation in a clear and understandable way,
4. training of the end User/Management in the use of the System and the interpretation of the output etc.,
5. helping and guiding the User/Management in developing the organisation and administrative structure of the data processing centre,
6. ensuring that all the required changes have been implemented and nothing is missing or is ambiguous.
7. It does not cover the activity of translating a Design into a machine executable form of a program or a form of a program that can be translated automatically to machine executable form. etc..

10. Does the Method contain a set of guidelines which can lead from the start of the project to the Implementation & Transition of the System?

Yes, JSD has clear guidelines on how to perform the two major steps i.e. the development of specifications and Implementation. For example JSD has the following specific rules and guidelines:

1. Functions should always be specified as process with long life times, and never as procedures.
2. A model process may be elaborated by introduction of variables, and of elementary operations. It should not be elaborated structurally in any way.
3. The Implementation should be done by using program inversion & other transformations for combining the sequential processes of the System Specifications.

11. Does the Method require any special training before using it?

Yes, the Method requires some training. The MJSL offers courses from time to time. The Method is not very complicated and can easily be understood by an average Designer.

12. What Methods, tools and support are used by the Method?

JSD may involve at the end, JSP and PDF, i.e. a program development facility, as a tool. These tools are not an integral part of JSD. These tools can be used at the

Implementation and the Transition Phases of the Life Cycle. The JSP is used for developing the Programs by developing data structure. PDF can be used for interactive production of the Program Design as a direct support for JSP techniques.

13. Does the Method offer clear and precise rules for choosing the Methods to be used in a particular environment?

No.

14. For what class of problem is the Method most suited?

JSD is concerned with the System for which subject matter is inherently sequential i.e. has time dimensions. This covers easily all types of data processing, process control, real time, embedded and all other sequential problems.

15. Is the Method language and machine dependent?

No, JSD is language and machine independent.

16. Does the Method assist with the project Management and Strategy?

No.

17. Is the Method based on principles which are in general:.

a. Structured,

b. Top-down,

C. Bottom-up,

d. Others?

Others. It has its own unique approach.

18. Does the Method use the V & V activities or something similar to them?

Yes, but not precisely and significantly.

19. To what extent deos the Method use the V & V activities?

They develop the System and validate it by going back to the User/Management and asking if the real world is really like the model they have developed.

20. Is the Method suitable for small/medium/large Systems?

It is suitable for small and medium problems. It can be used for Systems of realistic sizes and complexity.

21. Does the Method have communication facilities between the User/Management and the Analyst/Designer?

Not any specific one.

Note: There is no way expressed in their literature which can show how to communicate with the User/Management. Further there is no check to ensure that the problem has been understood clearly by the Designer.

22. What documentation facilities, tools does the Method have?

Each step of JSD produces some documentation as a by-product. For example structured diagrams are the by-product of entity/action and entity structure steps. The System Specification diagram is the output of the initial model step. The System Implementation diagram is the by product of the Implementation step. These documents can provide information about the System and exhibit the reasons for development decisions. For example the sequential process can be expressed by structure diagram as a tree structure of sequence, selection, and iteration. JSD has especial diagramatic and textual representations for such tree structures. The control flow of constructs of sequence, selection, and iteration are augmented by back tracking versions of the selection and the iteration constructs. The Quit statement transfers control to the end part of the iteration component or to the start of the second part of the selection. JSD does not have any specific tools for documentation purposes. The documentation can be produced by using the computer.

23. Does the Method use any symbols or notations, for expressing the outputs of the Phases, e.g. specifications, Design, etc.? If yes, what are they?

Yes, It has especial JSD notations for expressing sequence, selection and iteration components. It also has System Specification and Implementation diagrams.

24. Is the style of symbols or notations readable, can these symbols or notations lead to effective, rapid and unambiguous communications?

Yes. The style of symbols or notations are readable. All these are very clear, understandable and unambiguous. These symbols & notations are easy to draw.

25. Does the Method rely on highly symbolic formalisms?

Yes, but these symbols are also supported by some textual representation.

26. What experience is required by the Analyst/Designer in applying and using the Method?

The Analyst/Designer needs some formal experience to express process structure, understand process communications, etc.. The Analyst/Designer must be able to communicate with the User/Management and understand its environment. To achieve the above does not need especial skills.

27. Does the Method have provision for covering special cases such as:.

a.	Enhancement?	Yes,
b.	Maintenance?	Yes,
b.	Prototyping?	Yes,
c.	Quality Plan?	No,
d.	Search for Existing Available System?	No.

The activities concerned with Enhancement and Prototyping can be done by JSD. For example:

1. Functions of the System can be enhanced, if needed, by replacing the function process in the model of the real world.
2. Prototyping can be made by Implementing the System Specifications quickly and cheaply, without looking at its running cost.
3. Prototyping can be made by completing the model specifications and adding the functions to the model for experimentation.

In JSD Maintenance can be done easily in the cases where addition of a function process is needed in the model. The Maintenance can be done when System functions are affected without changing the model of the real world. When it is required to change an existing function process in the model or change the model itself, then it becomes difficult. "Changing the process during its life time is like changing a program during its execution" [p 352, 21].

28. What Prototyping tools/support does the Method have?

Not any particular one.

29. Does the Method incorporate a Quality Plan for a System?

No.

30. Does the Method use any Quality Metrics?

No.

31. Does the Method support the User/Management participation? If yes, to what extent?

JSD does not involve the User/Management except at the time of acceptence.

32. Has the Method some unique approaches? If yes, what are they?

JSD insists that the Designer should start, not by considering the required functions. JSD emphasises to demote the idea of functions. It insists to develop first a model of the real world with which the System is concerned. The action of the model reflects the action of the real world. The function of the System may be superimposed on the model. The basic form of model is a network of processes. Initially one process is dedicated for each separate independent entity. The process later can be scheduled by distributing the operating System functions among the process of the model System. The scheduling of the process can be wholly or partly determined at Design time.

33. Is the Method a combination or variant of other Methods? If yes, which ones?

No, JSD has come out of JSP. It is an enlargement of JSP, and has nearly the same principles.

A.4 LSDM Methodology

The following general questions concerned with a Methodology are applied to evaluate the strong and weak points to do adequate feature Analysis of LSDM Methodology [16], [71].

1. What are the objectives and scope of the Methodology?

Objectives:

The following are the objectives of LSDM.

a. to provide a standard frame-work for project development tasks, to increase productivity,

b. to provide means of identification, investigation & understanding of problems, User Requirements, constraints and working of the existing System,

c. to produce in a concise way the needs of the User/Management concerned with the System and to assist them in the selection of the correct development option,

d. to allow the User/Management to verify that their needs have been catered for,

e. to provide means and tools to achieve the Systems objectives,

f. to provide effective review procedures to eliminate possible errors and inconsistencies at an early stage of System development,

g. to document and Design a flexible System which meets the Requirements of the User/Management,

h. to provide the facilities which can enable the User/Management to review project progress,

i. to provide communication between the Analyst/Designer and the User/Management,

Note:

1. They have not shown how the standard frame-work can work in different environments. A frame-work without having facilities of by-passing Phases or activities can never work successfully in different environments and with different types of problems.
2. There is no statistics or evidence that the productivity of the System development task, or the number of Systems produced by using the LSDM has been or can be increased.
3. LSDM does not include in the frame-work means to identify the problems and Requirements in the environment where there exists no System.
4. The way to verify that the needs of the

User/Management have been catered correctly is crude and is not based on any systematic Management approach.

b. On the basis of the information available from their literature it is observed that the communication between the Analyst/Designer and the User/Management is not as effective and systematic as it is claimed. Only some document review procedures have been adopted for this purpose.

Scope:

LSDM is mainly concerned with the Analysis, Design Phases and some aspects of the Implementation activities. It is concerned with how to get a Design having clear definitions of files or data bases, programs and runflows. It does not directly address project planning and Management control of System development. LBMS project control Method or a set of standards can be applied to LSDM if desired so, to do the project planning & Management control.

2. What are the fundamental principles of the Methodology?

a. "the data structure of an application, and not the processing needs, should determine the structure of the system" [p4, 71],

b. "the development process should split into logical and physical phases. In the logical phases the emphasis must be on the detailed definition of requirements in terms of data structures, data flows and processing needs" [p4, 71],

c. provide a detailed guide to Physical Design control to create a System up to an acceptable level of performance.

3. What are the pre-requisites for the Methodology?

A detailed feasibility study report must be available to start the System development activities.

Note: Like SADT, JSD, etc., this Methodology also assumes that the preliminary work has been done and the feasibility study report is available. It is also assumed that the output of that preliminary work is reliable, reflects the true picture and can satisfy the needs of the User/Management. But this Methodology has an advantage over the other Methodologies in that it has provision to do the feasiblity study, if required by the Analyst/Designer.

4. What is the starting point of the Methodology?

Generally LSDM starts its activities from the Feasibility Study Report, but in the cases where a feasibility study is not done earlier and is thought to be essential, the LSDM may do this job by conducting the feasibility study. The frame work of this study is the same as that of the Analysis Phase or stage, with different levels of details and emphasis on particular steps.

Note: The criteria to decide whether or not to do the feasibility study depends on the project size and time scale and on how much preliminary work has been done which can help the development tasks.

5. What is the final point or step of the Methodology, and what is the deliverable of the Methodology?

The final point or step of the Methodology is the Physical Design stage. At the end of this stage files/DBMS definitions, program specifications, etc. are delivered. These are the refined set of database Design and program outlines.

6. What are the steps or Phases of the Methodology?

The following are the Phases of the Methodology. These Phases which are called stages by LSDM are further decomposed into a series of steps. These steps have a detailed list of the tasks or activities to perform.

a. Analysis of Systems operation and current problems,
b. Specification of System/User Requirements,
c. Selection of technical options,
d. Data Design,

e. Process Design,

f. Detailed Physical Design.

The first Phase of LSDM (called stage) starts with obtaining details about the working of the current existing System. These details are obtained in order to investigate & identify the problems and deficiencies in the System. In this Phase a logical, i.e. not physical, view of the current System is developed by using the existing DFDs or any similar information. From this logical view of the existing System, logical or ideal DFDs are developed. The results of this investigation and the logical DFDs are utilised to develop a list of existing problems and Requirements to enhance the System or remove the problems. This list of problems and Requirements is in narrative form and is called PRL. The PRL is reviewed in the light of the existing and the logical DFDs.

The second Phase or stage is concerned with taking the logical view of the existing System to do its Enhancement or replacement with a new System. In this Phase the actual Requirements of the User/Management are identified and its specifications are developed by utilising the output of the first Phase. A Possible set of System Requirements Solutions called SRSs is developed. The best and most suitable SRS is selected by the Analyst/Designer. Later the DFDs of the required System and Logical Data Structure i.e. LDS for the selected SRS are developed. These DFDs and LDS are then utilised to obtain models of Entity Life Histories. The ELH models identify the Events in a correct sequence which cause changes to the System's data. These models also specify the processing required to handle each Event. These ELH models are submitted to the User/Management for review and comments in the light of the Problem Requirement List i.e. PRL, LDS and Entity descriptions. The output of this Phase is called the 'Specification of Requirements' by LSDM.

The third Phase or stage starts with considering the specification of the Requirements and developing a list of technical options for the System. These options are determined from the DFDs of the required System; System Update Process Outlines i.e. SUPOs; Logical Data Structure; Problem Requirement List (i.e. PRL); Entity descriptions; Function catalogue; Function descriptions; etc.. These options offer different types of facilities, costs and System delivery time. Later the User/Management is asked to select the desired option. The Analyst/Designer may help the User/Management technically in the selection of the option, if so desired. The selected option is used later for the System development.

The output of Stage 3 i.e. the selected option for the System to be designed, LDS, and Entity description are utilised to obtain the Data Design and Process Design. The activities concerned with stages 4 and 5 overlap each other and therefore, can be performed simultaneously. A list of different required Functions which may be either enquiry or update Functions is prepared. This list is called an F.C i.e. Function Catalogue. The FC, the input/output description, and ELHs after preparation are checked to ensure that they are fully described and understood.

To develop Data Design, the input/output description is used and Relational Data Analysis is made. The result of Relational Data Analysis, Logical Data Structure and Entity description is utilised to create the Composite Logical Data Design called CLDD.

To develop Process Design, the Enquiry Function Catalogue and CLDD are utilised to create Enquiry Process Outlines called EPOs. Similarly the ELHs and CLDD are utilised to create Update Process Outlines called as UPOs. The EPOs and UPOs are used to validate CLDD.

The last Phase or stage is about the development of a Physical Design. The validated Composite Logical Data Design or CLDD i.e. the validated output of the Data Design stage & Process Design is the input to this Phase. The CLDD is automatically converted to the Physical Design when 1st cut Data Design and 1st cut Program Design rules are applied. The Physical Design rules are also applied to obtain the initial view of the physical Data organisation and Process. These rules also work as a control and tuning excercise in the development of a Physical Design to achieve the required performance objectives. The physical control helps in creating files/DBMS definitions, program specifications, operating schedules, development plans and manuals.

Note: The first three stages are concerned with the System Analysis tasks and the last three with the System Design. Little Implementation and no Transition tasks are covered by the Methodology.

How to define & select an Entity is not clear. What is a Process and how to define & represent it is not mentioned. How to obtain the System Process Outlines is also not mentioned but used during the 3rd. stage .
Niether have they defined the term Function nor mentioned how to represent it. It is also not mentioned in their literature how and from where to obtain the Function Catalogue which is required to be used in stage 3. A Further interesting point is that they again prepare a Function Catalogue i.e. FC [p 25-26, 16] during the Data Design and Process Design i.e. in stages 4 & 5. The same FC which has also been prepared in stage 3. It is also not clear what is the need and the basis for the selection of the FC.
7. Are these steps or Phases of the Methodology systematically proceduralised, or do they just give broad directions?

The Phases or steps of LSDM require a detailed set of rules and guidelines to be used at various points. These rules and guidelines, called by LSDM as techniques, seems to be systematically proceduralised but complicated and difficult to understand and use.
Note: In their literature not much information is given about these rules & techniques. It is, therefore not possible to further extend any comments on these techniques.
8. What Phases of the System Development Life Cycle does the Methodology cover?

The Methodology excludes the activities which are done by the Life Cycle under the Project Proposal Phase, except some which are concerned with the subphase Initial Investigation. LSDM investigates the most frequent problems which occur in the existing System.

The activities regarding the Strategy Phase are not covered by the Methodology.

The Analysis Phase of the Life Cycle is covered partially. For example:

1. the activities of the subphase 3.1 Analysis of Environment, is partially covered. LSDM covers this Phase by investigating and studying the factors which may effect the System environment i.e. the factors which are responsible for the occurence of incorrect data, creating inefficient Processes or Functions, etc.. The other aspects of the Analysis of Environment are ignored.
2. the activities of the subphase 3.2 Requirements Analysis is covered by LSDM in the sense that it makes an effort to identify the Requirements of the User/Management and prepare a list of the problem Requirements i.e. some thing which can be used as the basis of further development.
3. the activities of the subphase 3.3 Preparation of Requirement Specifications

is done by developing a set of specifications of Requirements. This is called System Requirement solutions or SRS. The SRS is a set of particular and precise statements of those things which are deemed to be needed and its solution. The best and most suitable solution is selected. This solution is used in transforming the Requirements into System Specifications and providing an initial model to exhibit the relevent information needed to develop a System Design. Further efforts are made to specify initial data, which might be needed by the System. LSDM achieves this by developing Logical Data Structure, DFDs and ELH models in the light of PRL.

4. the activities of subphase 3.4 Check for Consistency is done by reviewing and offering the comments on ELH models in the light of PRL.

5. the activities of subphase 3.5 Search for Existing Available System are not coverd by the Methodology.

6. the activities of the subphase 3.6 Feasible Analysis and Detailed Proposal are done during the 3rd stage of LSDM. This is achieved by developing feasible options called a list of alternate Implementation

technical options. These options indicate how the System can be developed with varying facilities & constraints. After careful study and consultation with the Analyst/Designer the best and most feasible option is selected by the User/Management.

The activities concerned with the System Specifications Phase of the Life Cycle are not done in a separate Phase or stage. These activities are done in the 2nd. and 3rd. stage and overlap each other.

The Design Phase of the Life Cycle is covered in the sense that it starts the Design activities by using the selected option, Logical Data structure and Entity description. All these are in fact System Specifications, expressed in technical terms. The Design Phase of the Life Cycle is covered in the following way:

1. the activities concerned with the subphase 5.1 the Planning and Decision of the Life Cycle are done by developing a list of required Functions, outlines of input/output descriptions, and the behaviour of entities & Process, etc.,
2. the subphase 5.2 the Conceptual Design is partly done by developing Data Design, Process Design, Composite Logical Data Design i.e. the detailed and final logical Data Design, Enquiry Process Outlines, Update Process Outlines,

3. the name and activities covered in the subphase 5.3 the Physical Design of the Life Cycle is nearly the same as covered by the LSDM. In this stage Physical Design rules are applied to get an initial view of the physical Data Organisation & Process, files/DBMS definitions, output details, program specifications, operating schedules, clerical routines, development plans, etc..

The Implementation Phase of the Life Cycle is covered partially by LSDM. It excludes all Implementation activities except the development of operating procedures and manuals, training the User/Management and to help & guide the User/Management in developing & organising the administrative structure of the organisation. LSDM has a separate Implementation package.

There is no evidence that LSDM covers the Transition Phase of the Life Cycle or anything similar.

Note: The procedure of reviewing and offering comments is based on personal observation & thinking. There is no criteria or metric which is applied by LSDM to achieve this purpose.

9. Does the Methodology give equal weight to the different Phases of the System Development Life Cycle?

No, the Methodology gives more emphasis to the Design Phase of the Life Cycle. Partial efforts are made to do the activities of the Analysis Phase. Little effort is made to do Implementation. Other Phases are ignored by LSDM.

10. Does the Methodology contain a set of guidelines which can lead from the start of the project to the Implementation & Transition of the System?

Yes, LSDM has a number of rules and guidelines which are integrated into its techniques. These techniques make it easier to perform the activities of the various Phases or the stages of the System development. For example:

a. LDST, the Logical Data Structuring Technique, this technique provides guidelines in creating data-structure and Entity Models.

b. DFDs i.e. data flow diagrams; these diagrams show how the information will flow from the real world to the System, through the System itself, and back to the real world. These diagrams also show how Processes & transactions fit together.

c. ELH, i.e. Entity Life Histories, this technique helps in developing the Entity Life History Model, identification and definition of Logical Process. This approach allows the System to reflect and behave like the real world.

d. TNF i.e. third normal form, this technique of data Analysis helps in understanding the data, their inter-dependencies and creating logical data groups i.e. record types to meet the Processing needs.

e 1st cut Data Design and 1st cut program Design rules, these rules automatically convert Logical Processes and Data organisations to Physical Design.

f. CLDD, Composit Logical Data Design, this is concerned with developing a detailed and final logical data model of the System. This model shows about data records (entities), data fields, data structure, data flows, access entry points, transactions, processing and relationships.

g. Physical Design rules, these rules are concerned with getting an initial view of the physical data organisation & Process. These rules also work as a control for meeting performance objectives during Physical Design.

Note: The details about the above guidelines or techniques are not available and therefore, it is difficult to say how successful and effective these are. The advantages & disadvantages and the problems with LSDM DFD, are the same as that of SASD DFD.

The concept of ELH seems to be similar to the JSD model of the real world. The concept of TNF is similar to that used by SASD. Little Implementation and no Transition tasks are covered by LSDM.

11. Does the Methodology require any special training before using it?

Yes, LSDM requires a formal training to understand the Methodology. The LBMS company has well established training facilities. The company offers training in the areas of Analysis and Design. For example for an experienced Analyst, Programmer or User they provide the 'Analyst skills workshop' which is about the use of LSDM. Also they have two more types of workshops about Structured Analysis and Structured System Design. These workshop give details and complete training about LSDM. Similarly they have a workshop for managers which can help them in evaluating various structured techniques, Strategies and Implementation approaches.

12. What Methods, tools and support are used by the Methodology?

In addition to the various techniques mentioned earlier the LSDM may use the LBMS Control, a project control support. This facility is available only for the User option.

13. Does the Methodology offer clear and precise rules for choosing the Methods to be used in a particular

environment?

No. The Methodology does however offer a facility which can help in choosing a Method or Strategy according to a particular environment.

14. For what class of problem is the Methodology most suited?

According to the authors, the Methodology is suitable for a wide class of problems such as on line Systems, database Systems using a number of products such as ADABAS, IMS, etc.. It is also claimed that LSDM can deal with large non database projects.

15. Is the Methodology language and machine dependent?

LSDM is not a Method of coding and is not language or machine dependent.

16. Does the Methodology assist with the project Management and Strategy?

The authors of LSDM assist in project Management and also in staffing. They have some techniques concerned with project Management and control. They offer courses on this topics mainly for Management personnel.

Note: How effective are these techniques is difficult to say, as there is no evidence, publication or report which can verify their claim about these techniques.

17. Is the Methodology based on, in general:.

a. Structured,

b. Top-down,

C. Bottom-up,

d. Others?

The Methodology follows a Top-down, Structured approach. It starts from a high level description of the physical or logical form of the System. This high level description shows a broad picture. This description is decomposed gradually and in a controlled manner, to get more detailed information about entities, their behaviour & relationship, and a representation of Events which effect an Entity in the System.

18. Does the Methodology use the V & V activities or something similar to them?

The Methodology does not use V & V activities as defined in the Life Cycle. However, the Methodology applies the V & V activities in a very limited way by making an effort in checking that important System logics have been adequately defined according to the Requirements. They also try to find out & cover the effects of un-common cases and error conditions.

19. To what extent deos the Methodology use the V & V

activities?

LSDM checks and proves the System logic during the Analysis and Design Phases or stages. This is done by identifying the Events that may change the System's data, finding the correct sequence of Events, providing a description of processing & error handling and validating the Requirements for each Event.

20. Is the Methodology suitable for small/medium/large Systems?

The Methodology seems to be concerned and suitable for various sizes of problems.

21. Does the Methodology have communication facilities between the User/Management and the Analyst/Designer?

Not particularly. However, the Analyst/Designer may consult the User/Management at various levels, if needed.

22. What documentation facilities, tools does the Methodology have?

The Methodology has a set of forms which are used for recording and reviewing. The Structured English and pictorial Methods are also used for this purpose. LSDM documentation is a part of and by-product of System development tasks.

23. Does the Methodology use any symbols or notations, for expressing the outputs of the Phases, e.g. specifications, Design, etc.? If yes, what are they?

Yes, the Methodology uses symbols or notations during the development of Entity Models, DFD and ELH. These Symbols and notations are nearly the same as those used by SASD & JSD. For example entities in the Entity Models are represented by a rectangular box with a name in it.

24. Is the style of symbols or notations readable, can these symbols or notations lead to effective, rapid and unambiguous communications?

Not much information is available on how effective and useful these Symbols are. However, as these are very similar to those of SASD and JSD, so their performance will be the same as those of SASD & JSD.

25. Does the Methodology rely highly on symbolic formalisms?

No.

26. What experience is required by the Analyst/Designer in applying and using the Methodology?

The Methodology can not be used with out having proper training. The LBMS and civil service college offer courses and a self instruction package. The pre-requsite of the LSDM or SSADM are experience of the Methodology and knowledge of the material in the package.

27. Does the Methodology have provision for covering special cases such as:.

a.	Enhancement.?	Yes,
b.	Maintenance.?	No,
b.	Prototyping.?	No,
c.	Quality Plan.?	No,
d.	Search for Existing Available System?	No.

Enhancement is done by taking the logical view of the current System and extending or replacing the new facilities which are required by the User/Management.

28. What Prototyping tools/support does the Methodology have?

The Methodology does not use the Prototyping concept.

29. Does the Methodology incorporate a Quality Plan for a System?

The Methodology does not use any specific Quality Plan, but it has regular and formalised checking procedures. These procedures can help to some extent in elimination of possible errors and improving the Quality of work in respect of its completeness and applicability.

30. Does the Methodology use any Quality Metrics?

The authors of LSDM claim that the Methodology provides ways to measure performance up to an acceptable performance level during the Design Phase.

Note: How they measure the performance and what is the acceptable performance level is not defined by the Methodology. In the absence of such level or metric it is ambiguous to determine the Quality of the System.

31. Does the Methodology support the User/Management participation? If yes, to what extent?

Yes, the Methodology supports the User involvement during all the System development activities. This helps in developing a System which can meet the needs of the User/Management. For example the different and partial views of data given by data flows, data structures and Events are needed to be combined to get a comprehensive picture of the System. This job needs a lot of interaction and involvement with the User/Management to get the anwsers to the questions which may arise from time to time.

32. Has the Methodology some unique approaches? If yes, what are they?

LSDM has two unique approaches. First, it develops a System by considering the three different views in parallel i.e. data flows, data structures and Events. Second, LSDM is based on a data-driven approach not on Functions.

Note: The LSDM uses data structure, information flows and Events as the basic components of the System development. These components are documented by Entity Models, DFDs and ELH models. Similarly JSD uses the data structure, Entity and their actions as the basic component. Structured diagrams, Entity structures and initial models are used to document these components. In fact LSDM uses the modified components of JSD and SASD.

33. Is the Methodology a combination or variant of other Methodologies? If yes, which ones?

LSDM is grown from LDBD i.e. LBMS database Design tool and from SSADM i.e. Structured System Analysis and Design Method. SSADM is a mandatory Methodology for the government departments in U.K. within its jurisdiction.

A.5 <u>MASCOT Method</u>

The following general questions concerned with a Method/Methodology are applied to evaluate the strong and weak points and to perform a feature Analysis of the MASCOT Method [22], [72], [73], [74], [75], [87], [88].

1. What are the objectives and scope of the Method?

Objectives:

The following are the objectives of MASCOT:

a. "to develop an integrated approach to the problem of software construction, operation and test" [p.2,22],

b. to provide a standard framework for software Maintenance throughout the life cycle of a software System,

b. to save overall on System lifetime costs, i.e. the costs of operation, maintenance and documentation etc.,

c. to provide Design visibility which is essential for effective management control,

d. to develop an approach to Design a real time software computer System in terms of a net-work of inter-connected parallel processes, which can be processed independently of particular hardware,

e. to provide a documentation scheme for MASCOT software which can effectively portray the System Design in MASCOT terms,

f. to provide a means of applying Quality assurance to the software,

Note:

a. There is no evidence that they have any standard framework for Maintenance, and the Method can result in an overall saving on the System lifetime costs.

b. "........MASCOT has been shown no worse than any other method of structuring and most users believe that the benefits of good structure (enhanced visibility, maintainability and integrity) are worth the cost" [75].

Scope:

MASCOT is only concerned with the Design and Implementation of real time software Systems. It is a **M**odular **A**pproach to **S**oftware **C**onstruction **O**peration and **T**est. "The scope of MASCOT2 is limited. It does not cope particularly well with big Systems or multiprocessor targets" [75].

Note:

a. The main activities concerned with the System development such as the Project Proposal, Strategy, Analysis, System Specifications are outside the scope of MASCOT.

b. The MASCOT is a programming approach. It is a programming style like other styles such as JSP, Modular programming etc..

c. From its limited scope it can easily be revealed that MASCOT is a programming Method rather a System development Method.

d. MASCOT2 is the present version of MASCOT, MASCOT3 is under progress.

2. What are the fundemental principles of the Method?

The following are the fundamental princples adopted by MASCOT.

a. To have a unified form of modularity throughout the total software.

b. To develop a test strategy to obtain a System of desired Quality, and apply it accordingly to test the modules or subsystems.

3. What are the pre-requsites for the Method?

A computer having filing, text-editing, compiling and diagnostic facilities is needed before starting the software development. This computer is called a 'host' computer. At the same time another computer having the facilities to carry out its operational tasks is needed. This computer in MASCOT terms is called a 'target' computer. In many cases the host and target are the same machine.

4. What is the starting point of the Method?

The starting point of MASCOT is the development of an overall software Design on the basis of the operational Requirements showing what the System must do for the User/Management.

Note: How and from where the operational Requirements should be collected is not mentioned. There is no check to show that the operational Requirements will really satisfy what the User/Management requires.

5. What is the final point or step of the Method, and what is the deliverable of the Method?

Implementation & test Phase is the final step of MASCOT. At the end a complete software System is delivered, which can be loaded on a target computer to run it under the control of the target's own MASCOT kernel.

6. What are the steps or Phases of the Method.

The following are the Phases of the Method.

a. Overall software Design,

b. Detailed Design,

c. Implementation & Test.

The first Phase of MASCOT starts by developing an overall software Design on the basis of the operational

Requirements. The Design is developed from co-operating parallel processes called activities and data areas (which may be either channels or pools). These activities and data areas are represented diagramatically by a network.

This network diagram is drawn only when the different Designer agree on the purpose and justification of each System element i.e. activities, channels and pools. The purpose & justification of these System elements are made on the basis of the operational Requirements and clear understanding of the purpose of software System. The determination of the purpose & justification of these System elements is done by re-examination iteratively. This network diagram is the basic structure of the System Design, and exists at only one level. The ACP network structure has no hierarichy and has no top or bottom. To understand the operation of the System the network diagram is decomposed into groups called subsystems.

After developing the network diagram the tasks of project planning, resources estimation, implementing and testing strategy are determined. This is done by considering the available man-power & time. For example, if the time and man-power allows the testing can be done extensively by testing each activity separately. Otherwise, group of activities or sub-systems can be tested. The network diagram is also used to determine the estimates of the required hardware, its configuration, capacity, cost, distribution of software between the processors etc..

In the cases where it is difficult or un-economical to obtain information about the hardware, Implementation Requirements, Design, resources needed, Implementation & test strategies etc., MASCOT develops a software Prototype. The ACP network diagram which depicts the System Design structure is used as the starting point of the Prototype. This network diagram is implemented on any MASCOT Kernel using a suitable high level language. The Prototype is refined, if needed. The final Prototype can be transferred to the target hardware System with little modification and no change in the Design.

Briefly the following may be the main deliverable of the Overall Design Phase 1.

a. A conceptual software Design in terms of an ACP network.

b. The purpose of each activity, channel & pool.

c. Proposals for Implementation & test strategies

d. Estimates of resources needed i.e. hardware, capacity, cost, man-power etc..

e. A software Prototype, in certain cases.

In the second Phase a detailed skeleton of the software modules developed in Phase 1 is made. The Test strategy is expanded to produce detailed manual and automatic support.

The ACP network, one of the outputs of Phase 1, is expressed in software technical terms. The root procedures are developed for the purpose of specifying the processing to be carried out by the activities. Each root procedure is designed as a normal single thread sequential unit of program, which can be programmed in any high level programming language. The channels and pools are used as parameters of the root procedures. The communication between activities is only allowed via those channels & pools which link activities in ACP network diagrams. This communication is synchronised by developing special access procedures. The access to data areas which belongs to channels or pools is made essentially through its access procedures. These access procedures hide from the activities both the internal detailed data structure of the channels & pools, and the use of the synchronisation primitives i.e. JOIN, WAIT, LEAVE, etc., provided by the Kernel. Later the initialisation and printing processes are developed.

The details of the data structure, access procedures, initialisation & printing procedures for each channel & pool form the Detailed Design framework.

Finally the details of test procedures are made and written in manuscript form. These test procedures are applied at various levels. For example, test procedures may be used to test channels, pools, activities, integration and acceptance.

The following are the main outputs of the Detailed Design Phase.

a. An approach to access the channels & pools.

b. Detailed skeleton of the software in manuscript form to be written in any suitable high level programming language.

c. Detailed approach of testing and test programs.

The third Phase is concerned with the Implementation & Testing tasks. In this Phase the outputs of the Phase 2 which are in terms of detailed Design Skeleton are converted into machine readable form, ready for compilation, editing, linking, loading, testing. This task is done by System software Construction facility. To construct software MASCOT generates a file of System Elements from source text modules. To do this the inter-connections of System Elements are removed from source text and inserted manually by creating sub-systems from System Elements using the Form facility. After the completion of the compilation work the testing of software starts. The testing is done according to the testing plan developed in Phase 1. It usually starts with the testing of individual modules developed in earlier Phases, testing them after integration and acceptence. The Implementation & Testing approach adopted by MASCOT differs from that of Design approach. The Implementation & Testing approach adopted is neither top-down nor bottom-up. In this approach activities are not called by an higher level routine. The following are the deliverables of the Implementation & test Phase.

a. Source text and compiled form of all software modules.

b. Test input data.

c. Test results.

Note:

a. From where and how to obtain information about the Requirement of the User/Mangement and System Specifications is not clear.

b. From where to obtain operational Requirements, information about processes i.e. activities, and data areas and knowledge about clear understanding of the purpose of the software is not clear.

c. Like JSD the MASCOT approach is biased towards processes rather than Procedures. But the use of processes (activities) in MASCOT differs from JSD. It differs in the sense that MASCOT functions may be programmed with no knowledge whatsoever of the context in which they will be used, apart from the essential data environments required for communication purposes. Where as in JSD functions are introduced at the last. In both Methodologies it is not clear how to determine, define and represent a function.

d. Like the JSD scheduler, MASCOT uses a Kernel for running several processes (activities) on a single processor.

e. What the criteria is for decomposing the System into a Sub-system is not clear.

f. The concept of global data is not used in

MASCOT.

g. How to obtain first the ACP network without having knowledge of its elements is not clear.

h. The concept of ACP is similar to the JSD process model which is an abstract description of the real world in terms of sequential processes.

i. The MASCOT kernel needs essentially to be developed by qualified and experienced staff. Otherwise the Method is so complicated that there is a great risk that the overheads and the size of the Kernel may increase dramatically.

J. The activities can communicate with each other only via IDA's i.e. channels & pools. This forces the activities to depend upon a set of IDA's.

K. The MASCOT software System is complex to develop. Perhaps this is the reason it is not popular and commonly used.

7. Are these steps or Phases of the Method systematically procedurised, or do they just give broad directions?

The MASCOT Phases give broad directions showing how to develop the System. Perhaps due to commercial reasons detailed procedures are only given when a contract is made with the developers i.e. RSRE.

8. What Phases of the System Development Life Cycle does the Method cover?

MASCOT covers only the Design and Implementation Phases of the Life Cycle. The tasks concerning the sub-phase 'Planning and Decision' of the Life Cycle such as decisions about the basic structure of the software Design are made in the 'Overall Software Design Phase', which is the first Phase of MASCOT. This is done by:

a. agreeing (iteratively) on the purpose & justification of each System Element i.e. channels, pool, and root procedures,
b. developing ACP network,
c. developing processes i.e. activities,
d. making project planning, resources estimation etc.,

The tasks concerned with sub-phases 'Conceptual Design' and 'Detailed Design' of the Life Cycle both are done in the 'Detailed Design Phase' of MASCOT. For example, the tasks concerning the 'Conceptual Design' i.e. the develoment of appropriate image of the actual Design and its information flow, processes, network, subsystems and their relationships etc. are achieved by:

a. expressing network in software terminology & subsystems, to obtain an appropriate image of actual Design,

b. developing modules (activities), and their interactions by specifying the processing to be carried out by the modules (activities) and developing root procedures, and their parameters i.e. channels & pools,

c. developing an approach for information flow and means of communication between processes i.e. access procedures,

The tasks concerned with the Detailed Design Phase i.e. the details of data format, data structure, data base, processes, modules, subsystems are made in the 'Detail Design Phase' of MASCOT. These are achieved, for example, by developing details of activities, channels & pools, root procedures, access procedures, initialisation and printing procedures, subsystems etc.. All these are done iteratively.

The tasks concerning the 'Implementation Phase' of the Life Cycle which is concerned with the use, manipulation, translation into code and debugging, of the software System frame or skeleton, training etc.. This is done in the 3rd. Phase of MASCOT named 'Implementation & Testing'. In this Phase the output of Phases 1 & 2 are converted into machine readable form by using a high level language, ready for compilation, editing, linking, loading, testing. The software Construction facility is used for this purpose. The debugging is done according to the testing plan or strategy developed during the earlier

Design Phases. The linking and integration is done by using a Form command which combines the System Elements i.e. channels, pools, root procedures. The training of the User/Management is done by having special training courses arranged by RSRE Ministry of Defence, U.K.

9. Does the Method give equal weight to the different Phases of the System Development Life Cycle?

No, more emphasis is given to the Implementation Phase of the Life Cycle.

10. Does the Method contain a set of guidelines which can lead from the start of the project to the Implementation & Transition of the System?

Yes, but not in detail, for example, MASCOT guides in selecting host computers which may be made by the facilities the Designer already posseses. It also helps in the selection of target computer and guides in making other important decisions such as to use a frozen or an evolutionary MASCOT, to access the needs of a multi-processor configuration, cost time etc..

11. Does the Method require any special training before using it?

Yes, the MASCOT requires to have proper training before using it. The MASCOT has a training course, run by

Computing Science Branch, Royal Military College of Science, U.K.. Most suppliers of MASCOT also run occasional courses of their own MASCOT.

12. What Methods, tools and support are used by the Method?

A translation tool named MORAL i.e. MASCOT Oriented Reliable Applications Language can be used to check for compatibility prior to compilation.

13. Does the Method offer clear and precise rules for choosing the Methods to be used in a particular environment?

No, the MASCOT is a Method which is suitable only for particular environments.

14. For what class of problem is the Method most suited?

The Method is suitable only for real time Systems.

15. Is the Method language and machine dependent.

No, but the MASCOT is developed with particular reference to the real time language i.e. CORAL-66. The Method can be used with other high level languages and with assemblers which can provide a convenient method of expressing the main structural items i.e. root procedures, channels, pools, and must be able to provide means of communication with the Kernel software. The software is not restricted to a particular hardware. The software developed by using MASCOT can be mapped on to many possible hardware configurations. The developers of MASCOT claim that the MASCOT System is flexible in the sense that it can be implemented on a bare machine and can provide a fundamental level of software, or it can be put on the top of an existing operating System. However the hardware configuration selected to run MASCOT can affect the Design of the kernel and programming of the subsystems software. In general the MASCOT simply uses hardware as a means of executing activities in the network.

16. Does the Method assist with the project Management and Strategy?

No.

17. Is the Method based on principles which are in general:

a. Structured,

b. Top-down,

C. Bottom-up,

d. Others?

"The Mascot design methodology belongs to the 'iterative top-down design family" [p.6,74]. The MASCOT is based on Top-down, Modular and Structured approaches. It decomposes a problem into set of smaller, manageable sub-problems by using the concept of Modular Structure. The results of this Modularity produces a hierarichy i.e. System--Subsystem--Activity, channel, pool(ACP) hierarichy.

Note:

a. The organisation of MASCOT software i.e. System into Sub-systems makes it possible to Design and implement the software System first on a large single computer and later on a network of computers. This makes it possible to study and investigate any software problems of distributed computer Systems, in the absence of hardware problems which may be due to distributed hardware problems.

b. The authors claim that the MASCOT Modular Structure has proved that it significantly reduces software integration time and provides re-usable programs modules. There is no support available for this claim. It is, therefore, difficult to say anything about it.

c. How to avoid the coupling between the modules and eliminate the ripple effects which may occur as a result of modifications of a module is not known. Further there is no evidence that MASCOT is making any consideration to have cohesion within a module. In the absence of these two measures there can be serious errors.

d. The work space or module environment is determined by considering the largest module size, by using the bottom-up approach. It is not easy to plan & manage the size of the module in advance. Further it is not an optimal way to calculate the size of the module environment.

e. The absence of cohesion and the presence of coupling may make it more difficult and less efficient to share computing time. Because in a Real time Systems where an interrupt can break the running of programs in an un-predictable manner it can effect the performance of other modules.

f. MASCOT allows only two levels of decompositions.

g. MASCOT Method essentially follows bottom-up or flat approach in which, firstly the activities are developed and secondly the sub-systems.

18. Does the Method use the V & V activities or something similar to them?

Yes, the MASCOT Method makes considerable effort to obtain Quality assurance.

19. To what extent does the Method use the V & V activities?

The Method applies V & V activities by developing a testing strategy during the Design Phase. This strategy is developed according to the resources i.e. time, money and man-power available. It also makes V & V activities by keeping standard documentations through out the software development.

20. Is the Method suitable for small/medium/large Systems?

The MASCOT is claimed to be suitable for large real time Systems for which it was mainly designed. In pratice it does not cope well with large Systems.

21. Does the Method have communication facilities between the User/Management and the Analyst/Designer.

Yes, the MASCOT has its own communication facilities for the use of the Designer and User/Management. MASCOT documentation is used for this purpose.

22. What documentation facilities, tools does the Method have?

The MASCOT has very specific and precise documentation standards which can be directly applicable to all MASCOT Systems. The MASCOT documentation standard is designed according to and can reflect the MASCOT approach. The MASCOT documentation consists of a set of graphical diagrams called 'Activity-Channel-Pool' i.e. ACP diagrams.

23. Does the Method use any symbols or notations, for expressing the outputs of the Phases, e.g. specifications, Design etc.? If yes, what are they?

Yes, MASCOT uses six different symbols & notations. These Symbols represent data arrow, sub-system or peripheral device, sub-system boundary, channel, pool and activity.

24. Is the style of symbols or notations readable, can these symbols or notations lead to effective, rapid and unambiguous communications?

The style of ACP symbols can show System Structure and flow of data. These notations can help effectively in communication during the System development Phases. The MASCOT Design documentation scheme has three parts, each showing the why, how, and where aspects of the System software. The why documents at the System level show System, physical System, processing description, interface description and test documents.

Similarly they show at subsystem level. At ACP level they show ACP diagram, processing description, channel, pool description and test documents. Further these Symbols are easy to draw.

25. Does the Method rely on highly symbolic formalisms?

Yes, MASCOT relies too much on ACP and therefore, pays more attention and consideration during the development of ACP diagrams, which are its basis. The laying out of ACP diagrams needs high skill and good experience, because these diagrams may be required to be refined several times at different stages.

26. What experience is required by the Analyst/Designer in applying and using the Method?

To develop a System using the MASCOT approach needs highly experience and qualified staff. Otherwise, there is a great risk that the resultant System may have more overheads and may not run in an optimal way.

27. Does the Method have provision for covering special cases such as:.

a.	Enhancement?	Yes,
b.	Maintenance?	Yes,
c.	Prototyping?	Yes,
c.	Quality Plan?	Yes,

d. Search for existing available System? No.

The authors of the MASCOT claim that MASCOT Method develops and implements a System in such a manner that its subsystems may be changed or modified at any time, if needed, while the System is in operation. In this way Enhancement can be made possible in MASCOT. The MASCOT Method can do the Maintenance tasks by changing or creating new sub-systems and introducing them into the working System without disturbing its working. The Maintenance tasks in MASCOT follow exactly the same as the initial integration of the System. About the Prototyping they claim "the MASCOT Method lends itself very well to software prototyping". [p.6, 72]. They develop the Prototype to obtain the resource estimate and outline Implementation of the ACP network. The MASCOT software Prototype essentially has the same structure as the proposed software. ACP network can be implemented on any MASCOT Kernel using any suitable high level language. The Protytype can be modified, if needed. The final Prototype is the transferred to the target hardware with minimum modifications.

Note: During the System Enhancement how they avoid the ripple effects and the problems of coupling between modules is not known.

28. What Prototyping tools/support does the Method have?

MASCOT does not use any tool/support for Prototyping.

29. Does the Method incorporate a Quality Plan for a System?

Yes, MASCOT has a Quality Assurance Plan. The MASCOT Quality Plan has six milestones and Quality control procedures. At each milestone the Quality Control procedures can be applied to ensure that Quality is maintained.

Note: There is no information available about the Quality Control procedures which are being applied.

30. Does the Method use any Quality Metrics?

No.

31. Does the Method support the User/Management participation? If yes, to what extent?

No.

32. Has the Method some unique approaches? If yes, what are they?

Yes, the following are the unique points in MASCOT.

a. The Method develops software first on a host computer and then implements it on a target computer.

b. It is possible to use a subset of MASCOT facilities and fix these facilities within the existing arrangements for compilation, loading etc.. For example, MASCOT allows the use of Kernel and synchronisation primitives excluding an interrupt handler.

c. MASCOT software can be implemented on a single processor computer or by a network of inter-connected processors. Having one processor for each process (activity). This approach is similar to one which is used by JSD.

33. Is the Method a combination or variant of other Methods? If yes, which ones?

Yes, the MASCOT was first developed during the 1970's. Later during the 1980's it was modified and produced as MASCOT2. Realising its weaknesses, since 1983, efforts are being made to rectify them and upgrade MASCOT to MASCOT3.

A.6 <u>SADT Methodology</u>

The following general questions concerned with a Method/Methodology are applied to evaluate the strong and weak points and to perform a feature Analysis of the SADT Methodology [11], [45], [89], [90].

1. What are the objectives and scope of the Methodology?

Objectives:

The following are the objectives of SADT:

a. to think about a problem in a structured way to do a systematic study of a problem,

b. to communicate and document the results of the Analysis and Design Phases by developing clear and precise diagrams supported by some text in a natural language,

c. to maintain and control the quality, accuracy and completness of the output i.e. the System,

d. to work and develop the System in a team spirit, to achieve effective cooperation and have better performance (results),

e. and to make the effort to manaage and assess the progress of a project.

Note: How to maintain and control the Quality, accuracy, and completness of the output i.e. the System Design model in a concrete and guaranteed manner is not clear. No Quality Plan or metric is available which is used by the Methodology for this purpose. Only some regular reviewing and approval procedures are available which can be applied to critical reviews. These continuous reviews and procedures can help to prevent drastic errors. But these cannot give concrete and guaranteed results to achieve a certain level of Quality.

Scope:

SADT covers the jobs which are concerned with System development, described by SADT as 'Functional Analysis' and Design. It deals with large and complex problems. When developing a new System the scope of SADT is the Analysis of the Requirements and functions, and then to Design and Implement according to the results of the Functional Analysis Phase, the functions and some other information. When SADT is dealing with an old existing System, its scope is concerned with the Analysis of the purposes, the functions which the old System performs and the new Requirements. It is also concerned with the study and documentation of how these purposes and functions are to be performed by the existing System i.e. to study their mechanism. The results in both these two cases are represented in a SADT model. This model shows the structure of the understanding gained.

2. What are the fundamental principles of the Methodology?

a. Analyse the problem in top-down, modular, hierarchical and structured fashion and express the results in depth. These results should be represented by the model(s).

b. Differentiating between the Functional model and Design model of a problem.

c. Represent the problem in an SADT model in terms of things and happenings. The model must show that both the aspects are properly related.

d. Use diagrammatic and natural language for documentation purpose.

e. The support team work in a disciplined manner to get better results.

3. What are the pre-requisites for the Methodology?

A statement of System Requirements, decisions and constraints about the System Design and information about functions are the pre-requisite information to be used as the input to the SADT System Development Project. SADT calls all these secondary requirements.

Note: SADT assumes that the secondary requirements, i.e. System Requirements, decisions, constraints, etc. have been derived from the User/Management Requirements and are available for use by SADT. Perhaps it is also assumed that these Requirements which are derived from what the User/Management 'wants' are correct, reliable, unambiguous and can serve the purpose which the User/Management wanted to have. But how and from where to get this information is not mentioned. Or perhaps they assume that the User/Management know better about their problems, its solution and their Requirements.

4. What is the starting point of the Methodology?

The SADT starts its work with a statement of System Requirements called secondary requirements. It starts with the most general or abstract description of the System to be produced. This description according to SADT is contained in a single module, represented by a box called a parent box.

5. What is the final point or step of the Methodology, and what is the deliverable of the Methodology?

The SADT System Design model is a major output of the Methodology. This model contains a set of diagrams. These diagrams show activity and data aspects of the System. This System Design model identifies the components of the software Systems and according to the authors can be implemented easily.

Note: How to Implement the System Design model is not specified by SADT.

6. What are the steps or phases of the Methodology?

The following are the Phases of the SADT:

a. Functional Analysis, (i.e. Analysis Phase)

b. Design,

c. Additional Analysis.

The Functional Analysis Phase of SADT is concerned with the process or steps of some aspects of the System Analysis and the development of the System Specifications. In this Phase efforts are made to know 'what' the System is supposed to do. The problem concerned with 'how' is desirably kept out from this Phase. However, in some cases where it is not possible to avoid 'how' this matter is also considered. The Input to this Phase is a statement of System Requirements, decisions and constraints about the System Design. This information is converted into System Specifications called Functional Specifications by SADT. These specifications are expressed by a model called the Functional model. This model consist of two sets of diagrams, called actigram and datagram. The first one describes the System in terms of its activities and the second one in terms of data. These two diagrams are always examined together to get both the aspects of a System. The order in which activities should occur and the sequence in which functions are to be performed is determined by these diagrams. The control and the mechanisms which effect and execute the functions are also identified. At the completion of these two diagrams efforts are made to check and make sure that all the elements of activity/data decomposition exists and are used consistently in its counter part of data/activity. This process is called A/D and D/A ties Verification, cross checking and error discovery. These diagrams and some other textual material showing the decisions and the reasoning used earlier are the output of the Functional Analysis Phase. During this Phase the Analyst sometimes may require to consult the User/Management for better understanding.

The SADT Design Phase follows the Functional Analysis Phase. The input to this Phase is the System Specifications (called Functional Specifications by SADT), and the Requirements and constraints imposed by the User/Management, which could not be considered or deferred at the Functional Analysis Phase. This Phase is concerned with 'how' the problem will be solved or executed. In this Phase details about solutions i.e. 'how' are developed. During this Phase various controls and mechanisms which effect and execute the functions are identified. The identification of the control and the mechanisms is purely on the basis of the experience of the Designer and the performance constraints. Like the Functional Analysis Phase the tasks of A/D and D/A ties Verification, cross checking and error discovery are also applied at the end of the Design Phase. The output of the Design Phase is the

SADT System Design model which consists of the actigrams and datagrams. This model, with additional textual support, can be used to a certain extent for the purpose of System Implementation. The model can recommend either a manual or automated System. To do the SADT Design Phase it is necessary to have a team of experienced Designer having extensive knowledge about the software techniques. This team must also know how to use these techniques effectively. SADT recommends to include in the team some persons who are also experts in the Implementation process.

The Additional Analysis Phase follows the Design Phase. In this Phase actions and Problems concerned with using, testing, training, transition, etc., are considered.

Note: There is no criteria known to us, which SADT uses to verify that the problem is fully understood. The Functional Analysis can give its results in more than one viewpoint. For example a Functional model can be developed from the User/Management point of view and from the Maintenance staff point of view. These two models from the same System Requirements will give different results. Moreover different authors produce different diagrams for the same problem from the same input. This situation gets worse when the same author produces different diagrams for the same problem from the same input when he/she is asked to produce them at some other time. Further it is impossible to make a one to one correspondence between the actigram and datagram. In fact to develop any sort of correspondence is extremely difficult. This may create conflicts.

The System development efforts in SADT start from specifying the functions. What and how the System will perform its functions is determined. The SADT Analyst/Designer use the available secondary information and selects the top most viewpoint of the problem. This top most viewpoint is decomposed into a hierarchy, to determine what and how the System will perform its functions. The top-down approach is adopted for this purpose. There is no criteria to select the top most viewpoint and to determine 'what is to be' or 'what exists'. All this is achieved on a guess or an idea of the Analyst/Designer which may lead towards ambiguous results. The decision of the top viewpoint of the System which is the highest level of the hierarchy may have widest possible consequences. It can condition everything that follows in the hierarchy. It may, therefore, be highly error prone, because it concerns with some thing i.e. the System itself which does not yet exist. The System viewpoint is placed at the very start of Functional Analysis, with out having done any proper Analysis of the problem, proposal and environment. This weak foundation may lead towards an unacceptable System. The Analyst/Designer who use such an approach perhaps have in their mind a System, developed by either way. They use the SADT Methodology only for describing and documenting the work which has already been done earlier.

There is no guarantee that the functions can be decomposed completely within the six levels to a point from where further decomposition is not needed. There are some problems which can't be expressed in the top-down manner. In such cases SADT may not work. The information about the System problem, the environment with which the System is concerned, and their Analysis are either ignored or covered partially and implicitly.
JSD and SADT both start their System development from the available information. JSD assumes that these available information are correct and can be used for the purpose of the System Design. SADT differs from JSD in the sense that it does not start the System development job by developing the model of the real world. SADT starts the System development efforts by analysing to some extent the available secondary Requirements and functions. Thus SADT tries to ensure that the problem has been understood to a reasonable extent before the details of the solution are finalised. In this particular aspect SADT is relatively better than JSD.

7. Are these steps or Phases of the Methodology systematically procedurised, or do they just give broad directions?

Yes, the first two main Phases of SADT are systematically procedurised. The last Phase i.e. Additional Analysis is not procedurised systematically. The following is a brief description of these Phases.

SADT starts from the general description of the System required to be produced. The Designer/Analyst chooses a viewpoint, and then from that viewpoint, identifies the main activity done or to be done by the System. This viewpoint is presented in a module, represented by a box. This module or box is further decomposed into a number of sub-modules or boxes, to represent the information in detail. This decomposition is done in the top-down, hierarchical manner. SADT provides clear rules for giving the numbers to the diagrams for the purpose of referencing within the model and documentation. Each diagram is either a summary diagram called a parent diagram or a detailing diagram called a child diagam. The Structured Analysis: a diagrammatic language is used for this purpose. This decomposition should not be less than three and more than six levels.

The SADT boxes within a model may represent either activities or data. If a model represents activities it will be called an actigram or otherwise a datagram. "Boxes in the actigram represent collections of related activities, not just monolithic actions. A box may perform various parts of its function under different circumstances, using different combinations of its inputs and controls and producing different outputs. These are called the different activations of the box" [p 4-5, 11]. The SADT rule for developing an actigram is that the first word in the actigram box must be a 'verb'. Each box can represent a major function of a parent box. The dominant activity is placed above, to the left side of the dependent activity, and its output is used as the control on the dependent activity.

In datagram boxes are some items of data and are represented by 'nouns'. In a datagram decomposition is based on class of data not on classes of functions. This diagram provides a concise reference to which activities can access each data item.

Each module and its relatives i.e. sub-modules must be connected by using inter connecting arrows. The SADT arrows show the interfaces. The arrows in an actigram represent the data of the System and the arrows of the datagram represent the activities. There are four specific rules which show how arrows can interact with the boxes. Each one for input, output, control or constraint and mechanism. The input arrows represent activity or data which is connected to the box of the actigram or datagram. This input is needed for transforming the purpose of the box. The output arrows represent activity or data which is coming out from the right hand side of a box of the actigram or datagram. This output is a result of the transformation of the input after passing through the box. The control arrows govern the way in which this transformation occurs. The mechanism arrows connect with the base of the box. These arrows indicate the process or device which performs the activity or stores the data i.e. the mechanisms which implement the purpose of the box. These may be like, sorts, data validators, queue handlers, lift, delivery van, car, etc..

The child or sub module is restricted to contain in it those elements that lie within the scope of its parents. The child module can't omit any elements which are part of the contents of the parent module. In the same way the child model can't add anything to the model. This provides a context which completely bounds the child.

Note: SADT claims that functions can be captured from actigrams and datagrams. How this can be achieved from the diagrams which themselves need to have complete knowledge about functions for its development, and how they define and determine the functions and other terms used in the System is not clear and is ambiguous. This approach can only be done when the Analyst/Designer have some pre concepts about the Design of the System and meaning of the terms which they want to use in the development of the System. In the absence of the above concepts and meaning the use of the terms function, Functional model, Functional Analysis, Functional Specifications etc. are ambiguous. The excessive use of the term function is ridiculous.
The arrows on SADT do not represent the flow of control or sequence, like in a flow chart. The SADT arrows represent constraints relationship among the boxes, in addition the criteria to determine a control is not clear.

8. What Phases of the System Development Life Cycle does the Methodology cover?

The Phases of SADT Methodology do not cover the whole span of the Life Cycle. The SADT Phases which are covered overlap with the Phases of the Life Cycle. For example SADT is not concerned with whatever is done under the Project Proposal, Strategy, Implementation, and Transition Phases. SADT is partly concernd with what is done by the Life Cycle in the Phases of Analysis, System Specifications and Design. For example SADT is not concerned with the Analysis of Environment, Analysis of Requirements, Search for the existing System, Feasibility Analysis & Detailed Proposals.

The Life Cycle term the Project Proposal is called by SADT 'primary requirements' for the System. SADT assumes that the primary requirements given by the User/Management have been examined by someone and are as readly available as the secondary requirements. These secondary requirements are used as the input to the Functional Analysis Phase to develop Functional Specifications of the System. Such jobs and actions are covered by the Life Cycle in the subphases preparation of Requirements Specification and System Specifications.

The Subphase Check for consistency is not performed specifically in the way it is covered in the Life Cycle. SADT checks only the consistency between data and activity aspects of the System Design. This is done to check any missing components etc..

The System Specifications Phase of the Life Cycle as mentioned earlier is partially covered by SADT. In the SADT Functional Analysis Phase efforts are made to find 'what' the System is supposed to do from secondary requirements. The objectives, constraints and functions to be performed are also evaluated from the secondary requirements, and are expressed by actigram and datagram.

The SADT Design Phase also develops some System Specifications, e.g. details about the functions, operations of the System, System boundaries and System interface are determined by this Design Phase.

The scope and the coverage of the Design Phase of the Life Cycle differs from that of the Design Phase of SADT. The SADT Design Phase is concerned with the limited Designing jobs as defined by the Life Cycle. For example it is concerned with 'how', i.e. to transform Functional Specifications into how to do it. This Phase converts the Functional Specifications into an SADT System Design model. SADT Design Phase develops a library of documents containing the terms, notations, etc.. It decomposes the problem in top-down, modular and hierarchical manner in order to study its processes, actions, activities, data, etc.. This decomposition is allowed up to a maximum of six levels etc.. These modules or sub-systems are connected by SADT arrows. The SADT does not contain any constructs like JSD sequence, selection, and iteration. Such constructs can help during Implementation and Program development.

9. Does the Methodology give equal weight to the different Phases of the System Development Life Cycle?

SADT does not give equal weight to the job concerned with the different Phases of the Life Cycle. It gives a little coverage on the Analysis & the Design

Phases and ignores the many other Phases. More emphasis is given on how to develop the System Specifications.

10. Does the Methodology contain a set of guidelines which can lead from the start of the project to the Implementation & Transition of the System?

It gives clear rules how to develop the SADT model at the various levels. For example there is a clear rule that a module should not be decomposed less than three and more than six sub-modules. Further SADT has rules about introducing the details into the model during its top-down decomposition. SADT does not cover Implementation and Transition tasks.

11. Does the Methodology require any special training before using it?

Yes. According to the authors of SADT a SADT User must receive formal class room instructions before using the Methodology. Further they emphasise the need to supplement this class room training with practical work under the supervision of an experienced SADT instructor to use the Methodology properly.

Note: Perhaps the above claim and their emphasis for practical training seems to be based on purely commercial grounds. The Methodology is simple and easy to use. The only thing which is needed is to have a clear concept of the various aspect of SADT with some previous Analysis and Design experience.

12. What Methods, tools and support are used by the Methodology?

SADT does not need to use essentially any tool or support during the development of a System. However, there are some tools which can be used as an extra help in certain types of applications to describe a System. For example SPECIF [89], a computer aided tool may be used for drawing SADT models, automatic controls & Verification of SADT basic rules, coherence of boxes, arrows and labels, program design aids, etc.. SAINT [90], a simulation language, designed for **S**ystem **A**nalysis of **I**ntegrated **N**etworks of **T**asks, etc., may also be used as a support.

13. Does the Methodology offer clear and precise rules for choosing the Methods to be used in a particular environment?

No.

14. For what class of problem is the Methodology most suited?

The SADT is suitable for a wide range of problems such as CAD, telecommunications, process control, data processing, etc..

15. Is the Methodology language and machine dependent?

No.

16. Does the Methodology assist with the project Management and Strategy?

Yes, but at limited level and it is achieved by managing the project in a disciplined and coordinated team work. The views of the project personnel are communicated at different steps of the Analysis & Design. Regular critical reviews are made in an organised way. The comments on the reviews are recorded for further examination by the senior Analyst/Designer. Also "there are two basic strategies used in reading a SADT model: the complete model may be explored (to as much depth of detail as necessary) to gain an understanding of the system modelled, or a specific detail may be studied to accomplish a specific goal (to design or program a software module, to understand the role of an individual in a people-implemented system, etc.)"[p. 4-14, 11].

17. Is the Methodology based on principles which are in general:.

a. Structured,

b. Top-down,

C. Bottom-up,

d. Others?

SADT is based on decomposing a problem in top-down, modular, hierarchical and structured fashion?

18. Does the Methodology use the V & V activities or something similar to them?

No.

19. To what extent deos the Methodology use the V & V activities?

The Methodology does not involve in any V & V activities.

20. Is the Methodology suitable for small/medium/large Systems?

The Methodology is suitable for large and complex problems. However, there is no reason why it should not be used for smaller and more simple problems.

21. Does the Methodology have communication facilities between the User/Management and the Analyst/Designer?

The Methodology has a well organised document handling and communication facility. The facility is limited only for the use of the Analyst/Designer. SADT has a libarian which does the record keeping, filing, and document distribution. This facility makes review point handling and communication easy & systematic.

Note: SADT frequently uses the words and terms without having giving explicit and clear definitions. For example the terms function, Functional model, Functional Specifications and Functional Analysis are used several times without specific definitions. The term Functional Analysis is used for the activities of System development [pl-1, 11]. There is no clear difference between the usage of terms of the SADT model and the Functional model. It is observed that, in the later publications of SADT the word function is carefully used.

22. What documentation facilities, tools does the Methodology have?

The Methodology uses a graphical language called SA to represent a SADT model. This language has the capacity to give details gradually, unambiguously and with out superfluous detail The SA has the capability to express module interfaces and also provides clear and unambiguous vocabulary. The language can be supplemented by natural language for further explanation, when needed. The document handling and communication facility given by SADT is not designed for the use of the User/Management.

The SADT communication and reviewing facilities also give some documentations as a by product. These documents can reveal the history of various decisions, and improvements made by the Analyst and Designer from time to time. For example actigrams and datagrams as a set, come with a supporting table of contents and a glossary.

23. Does the Methodology use any symbols or notations, for expressing the outputs of the Phases, e.g. specifications, Design etc.? If yes, what are they?

SADT uses 40 separate items of notations and conventions which are used to bound the subject matter. These notations express how the functions in the SADT model can be carried out by a mechanism. These notations also show how a single mechanism can perform related functions at several different places in the SADT model.

24. Is the style of symbols or notations readable, can these symbols or notations lead to effective, rapid and unambiguous communications?

Yes, the SA, a graphical language used by SADT has a well defined and systematic set of rules, symbols & notations which are easy to draw and read. But these symbols and notations may create a problem when required to be redrawn.

25. Does the Methodology rely on highly symbolic formalisms?

Yes. The success of SADT model depends on how the Symbols are used and drawn.

26. What experience is required by the Analyst/Designer in applying and using the Methodology?

The Analyst/Designer using SADT must have a clear concept of various aspects of the Methodology. The Analyst must clearly know what the System is required to do. The Designer must be personnel(s) with an intimate knowledge of Implementation techniques. The Methodology is in fact easy to understand and use. But the authors emphasise the need to have a formal training and attend the practical classes.

Note: A clear understanding according to the authors of SADT can't be achieved by studing the Methodology and the related documents. The authors emphasise the need to have formal classroom instructions along with practicals under the supervision of SADT instructors. This claim and the emphasis on formal training and practical sessions seems to be purely commercial.

27. Does the Methodology have provision for covering special cases such as:.

a.	Enhancement?	Yes,
b.	Maintenance?	yes,
c.	Prototyping?	No,
d.	Quality Plan?	No,
e.	Search for existing available System?	No.

The Enhancement and Maintenance can be done by identifying in the diagrams which activities and data structures are to be introduced, changed or modified. This involves almost all the Analysis & Design tasks of SADT, because it needs to redesign data structures and redraw diagrams. The above provision are not very effective in SADT.

28. What Prototyping tools/support does the Methodology have?

The Methodology does not have any Prototyping facility.

29. Does the Methodology incorporate a Quality Plan for a System?

No. The Methodology does not incorporate any Quality Plan.

30. Does the Methodology use any Quality Metrics?

No.

31. Does the Methodology support the User/Management participation? If yes, to what extent?

No. The Methodology does not involve the User/Management during the System development.

32. Has the Methodology some unique approaches? If yes, what are they?

SADT consciously support and emphasis to do the work of System Analysis and Design in a disciplined manner and with a team work spirit. It discourages Management to impose the burden of software development on one person. The Methodology has an unique and effective communication and critical reviewing facility which is essentially applied at the various levels during the developing of the SADT model. The output of SADT Methodology i.e. the System model can recommend Management to have either a manual or an automated System. An important aspect of SADT is that it involves the reader-author iteratively during the development of actigram and datagram. This approach improves the Quality of the SADT model.

33. Is the Methodology a combination or variant of other Methodologies? If yes, which ones?

No, SADT is not a combination or variant of other Methodologies.

A.7 SASD Methodology

The following questions are applied on Structured Analysis & Structured Design Methodology, to evaluate strong and weak points and to determine how they can be applied while using the life cycle. This Methodology is the combination of Structured Analysis (SA) and Structured Design (SD). It also called SASD [5], [12], [18], [20], [21], [61], [62].

1. What are the objectives and scope of the Methodology?

Objectives: To:

The Structured Analysis has the following objectives.

1. Provide tools and techniques to develop a clear pictorial model of an existing system,
2. Analyse, study, and understand the organistaion, environment and working of an existing System, to study the feasibility of implementing the proposal given by the User/Management stating what they want.
3. provide a pictorial model of a new System and other documents such as System Specifications or general Design of the

new System, for the information and satisfaction of the User/Management,

4. provide the input to develop a Structured System, which can accept maximum possible changeability, if needed by the Designer.

The Structured Design has the following objectives.

Resolve the complexity of a problem and develop Systems which are workable, understandable, reliable, flexible, efficient, maintainable and economical.

Note:

SA, perhaps assumes that organisations under study have a System. This may not be true, especially in new organisations. Further SA assumes that the User/Management can assess correctly their actual needs, the starting point of SA.

Scope:

The scope of the SA is primarily concerned with organisations where there is already an old computerised System, interfacing with manual Systems. It covers the Analysis of the old computerised System and develops details of data-flow, data-structure, process logic etc., called the structured Specifications. SA can also deal with organisations where there is a manual System.

Note:

SA is not concerned with the Enhancement of the existing System or improving its operations [p.240, 12].

The scope of the SD is to produce detailed System Specifications by partitioning and organising the Structured Specifications i.e. the output of SA, into black boxes called modules. Its scope is the extension of Modular and Top-down Design.

Note:

The development of System Specification activities as expressed in the Life Cycle are overlapping in SA & SD.

The partitioning or organising of the System Specifications in a top-down manner to get a hierarchy of modules based on functions is not an easy job, because it is a difficult and complicated activity to find out which functions should be in the System. This is especially true for a Designer who is not from the organisation. It will be difficult to know every detail about the functions and develop a perfect System. Such Systems may need frequent Maintenance. Jackson says "A system development method concentrating on functions is always liable to produce a system that is difficult to maintain" [p-8, 21].

The SD Method has developed a criteria for partitioning the problem. This criteria is not easily workable. In practice it is difficult for someone to know in advance about the System, its structure, and functions to be performed, i.e. to know about something which does not yet exist. Therefore the approach for partitioning the System may be error prone.

The foundation of SD theory is formal but is not based on mathematical concepts.

SD is a good Method for System development but in this Method there is no facility or check which can give a proof about the correctness of the System. Efforts are being made to modify the Structured chart to provide a mechanism for proving the correctness of the System. For example Chyou [62] has tried to modify the Structured chart by introducing some guarded commands in the basic construct of the Structured chart. The modified Structured chart called a Structured proof chart, is in its evolutionary stage.

2. What are the fundamental principles of the Methodology?

The fundamental principle of SA is to Develop the System Specifications, by analysing the old System and available information in a Top-down fashion by doing successive refinements.

The fundamental principles of SD are the following:

a. A large problem should be partitioned into a number of manageable parts by identifying: distinctive functions, or purposes to be performed. Ideally, each part called a module should perform a single function.

b. These modules should be as independent as possible, and must be solveable separately. In other words functional relationship between modules should be minimised. This criteria of relationship between modules is called coupling.

c. Each module should carry out a single problem related function. This means that intra-modular functional relationship or binding should be as high as possible. This criteria is called cohesion.

Note:

1. Even for an average size problem it is a very complicated and difficult task to find out its functions. The reason is that the functions are usually complicated. It is difficult to find out their details and implications on other functions. In a practical situation an element of processing may be functionally related in different ways and with different strengths with other elements. Therefore, to get minimum coupling and maximum cohesion is a difficult job. Also it is very difficult to see in advance the functional relatedness of one element with another.

2. In the functional approach the Maintenance problem can arise when the User/Management want to have some changes. These changes may effect the functional activities resulting in disruption of the Structure

of the System.

3. If some function is missing at an initial stage, it will remain missing, because there is no check to find such a missing function. In SASD, therefore, a check for completness can not be guranteed.

3. What are the pre-requisites for the Methodology?

For SA some proposal from the User/Management containing their wants is required to start the Analysis of the existing System.

For SD the Structured Specifications, the output of Structured Analysis is the pre-requisite.

4. What is the starting point of the Methodology?

The activities of SA start by investigating the System which may be manual or automated. This is done by studying the System, and other written documents supplied by the User/Management.

The activities of Structured Design starts from the statements of what the System is supposed to do i.e. the output of the Structured Analysis, called Structured Specifications or Functional Specifications. This document comprises of a DFD, Data-dictionary, Mini-specifications, Data-access-diagram etc..

Note:

One of the problems in using Functional

Specifications is that these specifications are usually liable to change. Some times these changes may be during the process of System development. This creates the problem of Maintenance which is an expensive and difficult activity.

5. What is the final point or step of the Methodology, and what is the deliverable of the Methodology?

The end product of SA is the Structured Specifications or the Functional Specifications. These specifications can be developed in more than one way, giving alternative Systems with varying facilities, benefits and cost. The specifications which are fully agreed by the User/Management are decided as the accepted Functional Specifications for the development of the System. These specifications are expressed by DFD, Data-dictionary, Decision-tree, Structured English etc..

The following are the end product of of SD:

a. The end product of the SD is a final version of the Structured Chart in which modules are described by rectangular boxes. The Chart depicts information about the functions of modules, control, input and output details. It also shows what will be done to that input. In some cases the modules are grouped into packages for the purpose of Implementation.

b. The Data Dictionary developed during the

Structured Analysis is also updated and delivered again at the end of the Design Phase.

6. What are the steps or Phases of the Methodology?

The following are the main steps of the SA:

a. Do an initial study of the existing System or evaluate the request made by the User/Management, and write a report thereof,
b. Perform a detailed study of the initial report, study of facts, problems, limitations of the old System and submit the remedy for the problems,
c. Study the old System Specifications, if they exist, and consult the User/Management,
d. Produce an overall System data-flow and express it in DFD,
e. Develop detailed data-flows and express them in a detailed DFD,
f. Define the process logic and function logic by using the SA tools such as Structured English, Decision trees, etc.,
g. Break each component process, if necessary, to further level of detail,
h. Break down each logical function, if possible, and get more detailed DFD,

i. Develop data elements, data structures and data stores and revise all these, if necessary,

j. Develop a data dictionary containing in alphabetical order the contents of each data store & data element of which they are composed.

Following are the main steps of the SD.

a. Derive Structure chart,

b. Refine Structure chart,

c. Specify modules,

d. Pack the modules.

7. Are these steps or Phases of the Methodology systematically proceduralised, or do they just give broad directions?

SA does not provide step by step clear & precise guidelines showing how to proceed and achieve the goals.

Note: The tools provided by SA are better and helpful.

Yes, the Structured Design gives clear and systematic guide lines and procedures. These guidelines specify how to achieve the objectives i.e. coupling and cohesions. It also directs how to do the factoring, System's shape, fan-in, fan-out, packaging, error reporting etc. to Design a maintainable System.

Note:

These guidelines are not enough for an

inexperienced Designer. These guidelines can be helpful to one who has previous sound experience in developing a System by using SD.

8. What Phases of the System Development Life Cycle does the Methodology cover?

The activities of SA in general are not covered in the way they have been defined in the Phases of the Life Cycle. Following is a brief description showing how the Life Cycle can absorb the activities of SA in its Phases, and what activities are not covered by SA.

SA covers some of the activities of the Life Cycle specified in the subphase 1.1 i.e. the 'Study of Proposal'. SA deals with the proposals which are concerned with the development of new Systems. It eliminates those proposals which are concerned with the Enhancement of the existing System. The subphase the 'Study of the Proposal', has a wider scope of activities than that covered by SA because it can deal with all types of proposals whether concerned with Enhancement or having a new System. In SA there is no step which can find out at an initial level what the User/Management actually 'need'. SA does not make any effort to find out the 'needs'. The approach used by SA is perhaps purely of a commercial nature, in their own interest.

SA covers the subphase 1.2 i.e. the 'Initial Investigation', and starts the investigation, only when the proposal from the User/Management is to develop a new

System. The justification for having a new System is made on the personal assessment of the User/Management. The purpose of this step is not to find out the actual needs, but to start the development on the basis of their proposal.

The subphase 1.3 i.e. the 'Initial Feasibility Study', whose basic purpose is to find out the possible ways to implement the results of the Initial Investigation. SA, analyses and understands the present existing System, to achieve this purpose. The tools of SA are used for this purpose. SA gives a report of this Initial Feasibility Study. This report contains estimated costs, potential benefits, tentative completion dates, functions of the new System etc..

SA does not cover the Phase 2 i.e. the Strategy Phase of the Life Cycle.

SA covers few of the activities mentioned in the subphase 3.1 i.e. the'Analysis of Environment'. It tries to do the Analysis of Environment by discussions with the User/Management and reviewing the available documents, if there are any. The SA covers the study about the role & duties of the User/Management, and their internal hierarchy in the System. It develops the System boundaries, dictionary of the terminologies used in the existing System, nature of input/output data, functions to be performed, logic of process etc.. This information is essential to understand the System's environment.

The activities of the subphase 3.2 i.e. the 'Requirement Analysis', in SA are done by breaking the complex Requirements given by the User/Mananagement into its components. This is achieved by reviewing the plan given by the User/Management. SA covers the study of managerial, financial and social Requirements.

The activities of subphase 3.3 i.e. the 'Requirements Specification' are performed in SA by deciding what the System or part of it, is required to do. This is done by producing an overall System data-flow, deciding functions to be included, or omitted etc.. All these activities are done by studying the System Specifications of the existing System, consulting the User/Management, deriving objectives for the new System and developing a DFD of the existing System. The Requirements Specification are expressed by producing an overall DFD.

The activities covered in the subphases 3.4 and 3.5 i.e. 'Check for Consistency' and 'Search for Existing Available System' are not covered by SA.

Most of The activities mentioned in the subphase 3.5 i.e. the 'Feasibility Analysis and Detailed Proposals' are not covered by SA. The activity of cost benefit Analysis for the new System is done by producing a statement of possible revenue, improved service etc.. The activities concerning information about the essential changes in the policies, environment, status of personnel, organizational changes and effect thereof, may be performed by defining the User community of the new System.

SA produces several alternatives with differing sets of benefits and costs in the form of a report. The tools of SA help a lot in presenting such alternate proposed models. These alternate models give various versions of the Requirements Specification etc.. These proposal are submited to the User/Management for approval.

Some of the activities covered in the Phase 4 i.e. 'System Specifications' are covered by SA. For example the activities such as defining the System boundaries, System interface etc. are done in SA by deciding which functions are to be considered as part of the System Study and which ones are not to be covered. Further details about the process logic, function logic and operations of the System are achieved by breaking down the component process and logical functions etc.. All these are achieved by using different tools of SA. For example SA produces a logical model of the new System by using the DFD. This DFD can show some relevent specifications for the new System such as new data-flow, new data-structure, new transactions etc..

The Structured Design mainly covers all the activities of the Design Phase of the Life Cycle. It also covers certain activities which are covered in the System Specifications Phase of the Cycle. For example what data and actions will be required to perform a function. Another example may be that it specifies details about functions, modules and their interactions. However the SD does not cover how to represent the output of the Analysis results in technical terms. For example it does not specify how to fulfil the terminology and pragmatics of the application area into the terminology and pragmatics of computing. All this can only be obtained by the experience of the Analyst/Designer who has worked in the similar fields. Another example maybe that it does not show how to convert the User's Requirement that a System should be reliable to a required extent. In other words it does not say how to have a reliability metric. It also does not show explicitly how to test the Design.

Structure Design uses the Planning and Decision activities of Design Phase by developing a network representation of System components showing how data will flow, and what process are needed etc..

SD uses the Conceptual Design activities of the Design Phase by developing Structured charts from DFD. It is possible to develop a number of alternative Conceptual Designs based on these charts.

9. Does the Methodology give equal weight to the different Phases of the System Development Life Cycle?

SASD, i.e. Structered Analysis and Structured Design covers Analysis, System Specifications, Design and Implementation Phases of the life Cycle.

Structure Analysis overlaps some activities of System Specifications Phase, and also Structure Design does the same. A part of the activities concerned with the Implementation Phase are also covered by Structured Design.

SASD does not give equal weight to the Phases of the Life Cycle.

10. Does the Methodology contain à set of guidelines which can lead from the start of the project to the Implementation & Transition of the System?

SA does not give any particular set of guidelines or Analysis strategies. It only gives general information showing how to do the Analysis by using various tools, mainly in commercial organisations.

The SD has two major sets of guidelines or Design strategies. These strategies are called transform and transaction Analysis. Transform Analysis is a special case of of Top-down approach. It guides how to start from the output of Structured Analysis and get the Structured Design. This Strategy identifies essential or major functions which take major input data streams and create major output streams by transformations. This Strategy uses DFDs develops the structured chart for the Design. It has five clear steps which can give a transform centered Design. For the commercial environments where Systems are mainly concerned with processing of transactions the Structure Design suggests the use of a transaction Analysis Strategy to develop a Structure chart. In this Strategy the input data stream is split into several discrete output sub-streams. This Strategy can pass all transactions to subordinate transaction modules for processing. This approach a gives transaction oriented System.

Note:

The techniques mentioned above have certain limitations. For example these can give a correct Design, but its perfection cannot be guaranteed. Especially transform Analysis cannot be helpful for a big problem having a complicated DFD along with many efferent and afferent streams. The Transaction Analysis technique is offered for use in solving big problems. This can be done by dividing complicated DFD into smaller DFDs. However transaction Analysis has problems itself. For example transactions are problem oriented and it is difficult to draw structure charts especially in the large commercial organisations where different types of transactions are dealt with. These transactions are usually processed totally differently. Further in practice it is required to map the DFD into a particular Modular structure which is in practice a difficult job to do the first time. Only an experienced Designer who has some previous knowledge can do this. These DFDs have limitations in that they cannot be implemented directly by using a machine, because they are not in algorithmic form.

11. Does the Methodology require any special training before using it?

For SA & SD an experienced Programmer, Analyst or Designer with clear understanding about the Analysis & Design activities is needed. Usually they do not require any special qualification.

12. What Methods, tools and support are used by the Methodology?

SA uses the following tool and support during the Design activities.

a. **Data Flow Diagram:** They use it to draw a pictorial model of a System. The DFD shows Systems boundaries, sources and destinations of data. It identifies and names the functions, data elements and data stores. A DFD uses four Symbols.

Note: A DFD does not depict any information about timing considerations. It is therefore not suitable for a larger class of problems which are heavily dependent on time ordering. This tool can not show how the System can be implemented.

b. **Decision Tree:** This tool is used to express and present logic, structure and policies of a process. The tool is suitable where a process has complex branches. Because the branches of the tree can correspond to each of the logical possiblities.

Note: This tool can only be used effectively in a situation where the number of actions is small. It has limitations in that it can not show how to do calculations and to show any instruction to achieve the decision. Further there is a disadvantage that it is not machine readable.

c. **Data Dictionary:** This tool provides a structured place for keeping contents & details of data flows, data stores and processes. A Data Dictionary may be automated or manual. It is a sort of directory which contains information about a project. It is like a central store. A Data Dictionary can give various different types of outputs such as a full listing (details about data flows, data structures, processes etc.), summary listings (expressing all entries in an arranged and short form), etc..

d. **Decision Table:** This is another tool for expressing the process logic. This tool can give a more compact representation than any other tool can give.

Note: This tool can be more helpful in a process where the number of actions or alternative solutions are large and branches are complex like a decision tree.

The problem with a Decision Table occurs when there is a need to change the rules. This tool has further limitations that the User/Management can't understand and verify the Decision Table unless they are familiar with them.

e. **Pseudo Code:** It specifies program logic by using conventions of structured English. This language is free from the detailed syntax of any particular programming language. Therefore, this language can be used in any particular environment. Pseudo Code shows how to initialize/terminate, read/write the files. This code can give detailed Program Design.

f. **Structured English:** This tool is suitable for the process where some operations and few decisions are required. It uses logical constructs to express the logic of a process. In Structured English instructions, decisions and loops etc. can be written in terms of sequential instructions, case instructions, decision instructions and repetitions instructions. The terms defined in the data-dictionary are used if necessary, to make the language compact and clear. Thus by using this tool a logic process can be expressed very close to a computer program.

Note: Generally in SA the specifications about reading and writing the physical files are not expressed in Structured English.

SD uses the following tools and support during the Design activities.

a. **Structured Chart:** This tool can show an outer picture of the modules in a System and their relationship with each other.

Note:

1. For a large problem it might be necessary to have say, 500 modules. In such cases it

will be difficult to draw and develop a Structure chart, to get the Structure Design. To solve such problems it may be proposed to develop major subsystems. The problems in this approach may be how to get the input for the subsystems, have uncoupled or independent sub DFDs, and avoid duplication of modules.

2. Also the Structure chart can only give a time independent model of a System in which the modules are arranged arbitrarily. It is therefore incapable of showing intrinsic implications for execution order. Therefore, it is not very successful for sequential and time ordering problems.

3. Furthermore the Structure Chart does not show procedural details i.e. 'how' things like loops, decisions, initialisations, termination sequences of calls etc., can take place. It is unable to depict the internal data of modules.

4. The Structured Charts which are drawn manually are error prone. It is difficult to amend and verify them. The development and Maintenance of Structured Charts may result in a major effort, sometimes it may require complete re-drawing. Checking for completness and consistency between

different charts is not easy especially in the case of big Systems.

b. Psedo-code: This tool is mainly used for module specification and its Maintenance. It is also used as a support for Modular programming.

Note: This tool has an advantage that it is not language dependent. It differs from Structured English in the sense that it is only used by the Designer and the Programmer. It is very similar to the actual code, more closer than Structured English. But the Pseudo-code has problems that it takes more documentation than the actual code itself. It may affect the programming style. This code can lead towards ambiguous results because of its style, i.e. lack of punctuation. The code is incapable of specifying input/output media.
The tools of Structured Analysis such as DFD, Data Dictionary, Structured English and Data Access Diagram are also used closely in developing Structured Design.

13. Does the Methodology offer clear and precise rules for choosing the Methods to be used in a particular environment?

No, SA does not offer any such Method or approach for choosing a Method for a particular environment. Only the choice of a particular tool can be made according to the type of a problem.

Yes, SD offers two different techniques suitable for different environments. For example for commercial and real-time Systems SD recommends the transaction Analysis Strategy. For other small problems transform Analysis is suitable

14. For what class of problem is the Methodology most suited?

SA is developed mainly for commercial data processing Systems, and therefore is suitable for such types of problems. There is no evidence that it is also suitable for other environments. The nature of the SA approach reveals that perhaps it will not be suitable for the problems which are mainly concerned with time ordering and are of a mathematical nature.

SD is suitable for commercial problems. However, the developer of SD Method claims that it can be used to Design any computer System e.g., compiler, operating System, etc.. Again it is not very successful for problems which are heavily concerned with time ordering and are of a mathematical nature.

15. Is the Methodology language and machine dependent?

No, both the SA and SD are language and machine independent.

16. Does the Methodology assist with the project Management and development Strategy?

No, both the SA and SD have no such facilities.

17. Is the Methodology based on principles which are in general:.

a. Structured,

b. Top-down,

C. Bottom-up,

d. Others?

The SA uses a Top-down approach by doing successive iterations, to achieve better output of the Analysis activities.

The SD is based on modified concepts of Top-down and Modular Methods. It does not contradict anything in the Top-down or Modular approaches. It complements the guidelines which Top-down lacks by adding certain features to it. For example SD uses factoring to achieve advantages of Top-down decomposition.

18. Does the Methodology use the V & V activities or something similar them?

No, the SA and SD do not use the V & V activities.

19. To what extent does the Methodology use the V & V activities?

It does not use the V & V activities.

20. Is the Methodology suitable for small/medium/large Systems?

The SASD Methodology can be applied equally to all different sizes of commercial problems.

21. Does the Methodology have communication facilities between the User/Management and the Analyst/Designer?

The SASD Methodology does not have any particular communication facilities, between the User/Management and the Analyst/Designer. However, the output of their tools, can be used for communication purpose. But for this the User/Management must have a good knowledge of Systems Analysis and Design.

22. What documentation facilities, tools, etc. does the Methodology have?

The DFD, Data-dictionary, Decision-table, Decision-tree, Pseudo-code, Structured English which are the by product of the Analysis activities are used for the purpose of documentation.

The structure chart, pseudo-code and

Data-dictionary etc. which are the by-products of the Design activities are used, to some extent to show and document the structure of the System.

23. Does the Methodology use any symbols or notations, for expressing the outputs of the Phases, e.g. specifications, Design, etc.? If yes, what are they?

The SASD Methodology has a well defined and clear set of symbols & notations for expressing the source, sink, data-flow, data-store, and other structures of a System. For example a rectangular box with its name inside is used to represent a module.
Note: Some symbols used by SA differ from that used by SD. For example In SA a process is represented by an upright rectangle having rounded corners and divided into three sections, showing details of the process. Whereas in SD a small circle or bubble is used showing its name. The symbol used by SA is better than that of the SD, because it can give more details. Source or sink i.e. a terminator, is represented in SA by a double square whereas a rectangle with a name inside is used for the same purpose by SD.

SD uses a rectangle to represent a module. The same symbol in JSD is used to represent an entity, action and process. etc..

24. Is the style of symbols or notations readable, can these symbols or notations lead to effective, rapid and unambiguous communications?

Yes, these notations are clearly readable, easy to draw and can give unambiguous documentation for the use of the Analyst/Designer. They communicate with each other in a clear manner. These notations can be arranged to show the partitioning of a System and the hierarchy of the modules. The Structure chart has some limitations. For example it can not show the temporal order in which modules will be activated. It is unable to show the sequence of calls to other modules, loops, statements etc..

25. Does the Methodology rely on highly symbolic formalisms?

No.

26. What experience is required by the Analyst/Designer in applying and using the Methodology?

To use SA for analysing a System or studying a proposal requires an Analyst or a group of Analysts with some experience. The level of experience required is based on the size of the problem to be solved and its environment. The SA is in general easy to apply.

To Design a System based on SD needs essentially a group of experienced Designer who have worked before on similar environments. The Design approach is such that an inexperienced Designer, in a complex problem will not be able to apply the SD effectively and efficiently.

27. Does the Methodology have provision for covering special cases such as:.

		SA	SD
a.	Enhancement?	No.	Yes.
b.	Maintenance?	No.	Yes.
c.	Prototyping?	No.	No.
d.	Quality Plan?	No.	No.
e.	Search for Existing Available System?	No.	No.

The SD can do Enhancement and Maintenance tasks by modifying & changing the modules. This can be achieved by using the Structure Chart which is developed by the concept of dividing the problem into modules. These modules are developed by making efforts to maximise the Cohesion and minimise the Coupling. This can help in developing a System which may be easy to enhance and maintain.

28. What Prototyping tools/support does the Methodology have?

The SASD does not use the concept of the Prototyping.

29. Can the Methodology incorporate a Quality Plan for a System?

The SA does not use any Quality Plan. However, SD uses, Indirectly the Quality Plan by using the criteria of how to develop modules.

30. Does the Methodology use any Quality Metrics?

SA does not use any Quality Metric, but in SD the concept of coupling and cohesion can be used as a measure of Quality of a System.

31. Does the Methodology support the User/Management participation? If yes, to what extent?

SA does not involve actively the User/Management in System development. It only consults the User/Management during the study of the System by conducting interviews.

SD does not involve the User/Management during the System development activities.

32. Has the Methodology some unique approaches? If yes, what are they?

SA does not use any unique approach. But the concepts of coupling & cohesion, transaction & transform analysis used by SD for achieving better Quality of modules is of a unique nature.

33. Is the Methodology a combination or variant of other Methodologies? If yes, which ones?

Yes, the SASD is the combination of Structured Analysis and Structured Design.

APPENDIX B

B.1 Introduction

In this appendix the important key words or terminologies used or defined by the selected Methods/Methodologies are studied. The frequently used key words or terminologies are examined and compared with the definitions given in the IEEE glossary, the glossary of terms developed in this thesis and the way these have been used or defined by these Methods/Methodologies. In section B.2 the key words or terminologies defined or used by the Methods/Methodologies and by the Life Cycle are examined and compared with each other. In sections B.3 to B.8 the key words or terminologies are examined and compared with the definition given in the IEEE standard glossary. A set of 2 questions is applied for this purpose. The order in which these Methods/Methodologies are placed is alphabetical.

B.2 Definitions

This section describes how the following key words are defined or used by the Methods/Methodologies under study and by the Life Cycle, if defined in different & specific ways?

Action:

The term Action is not used by LSDM, MASCOT and ETHICS Methods/Methodologies.

JSD defines the term action as an event in which one or more entities participate by performing or suffering an action e.g. PLACE, DELAY, STOP, etc.. Actions in JSD take place at a point in time, not over a period of time, and these actions cannot be decomposed further.

SADT uses the term activity as a near synonym to the JSD Action. The SADT Activities are used for the happenings e.g. CONTROL FLIGHT, MANAGE RESOURCES, PRODUCE, etc.. These activities are performed by entities like men, machines, ... computers etc..

In SASD the SA uses the term action to refer to an imperative phrase such as STOP, PAYMENT, PRINT NAMES, etc..

Note: The terms Action, Activity are used by JSD, SADT and SASD for similar purposes i.e. for happenings or events performed by entities and are represented by imperative verbs.

Activity:

The term Activity is not defined or used by JSD, LSDM and ETHICS. MASCOT uses the term Activity for process, a part of program which may be developed independently. A MASCOT Activity is a unit of DP which executes identifiable tasks within a System. SADT uses the term Activity quite differently from the MASCOT term. It uses the term for happenings, such as CONTROL FLIGHT, ADD ALLOWANCES. The synonym of MASCOT Activity in JSD & SASD may be near to Process and Module respectively.

Analysis:

The term Analysis is used in the Life Cycle for the purpose of the systematic study and examination of needs, (obtained in an earlier Phase), in order to find, determine and identify the causes of the problems and the Requirements of the User/Management. The results of this Analysis is represented in detailed proposal(s) together with the feasibility report for developing the System Specifications.

In JSD and MASCOT the word Analysis is not used as a term. The term is also not defined explicitly by LSDM, ETHICS, SADT and SASD. LSDM uses it for obtaining the details of the existing System to investigate and identify the problems & deficiencies. ETHICS uses the term for examining the preliminary key objectives, Functions, identification of variance, examination of existing task structure and work flow. SADT uses a different term Functional Analysis for analysing the System Requirements and understanding what the system is supposed to do. The term Analysis is used by SASD for the jobs involving the study of the User's current System (if there is one), interviewing the end user, groups, and clerks, to find out how they are presently working, document User's needs by using various tools and ensuring that the User's statement of problems is not incomplete, redundant, or contradictory.

Note:

The basic term Analysis is not uniquely defined and covered by the Methods/Methodologies. It is assumed by LSDM and ETHICS that a System exists and its User/Management correctly know about their problems, needs and wants, which may not always be true. Further LSDM and SADT cover the limited aspects of the Analysis tasks.

Analyst:

The term Analyst is defined by the Life Cycle as a person who identifies the working environment, talks and works with the User and their Management, in order to discover, study and analyse their needs & wants; and prepares proposals. The term Analyst may be used as singular and plural and for male or female.

JSD and MASCOT are not concerned with the Analysis tasks and hence the term Analyst or System Analyst is not used. LSDM, ETHICS and SADT have not defined the term Analyst or System Analyst, but the term is used. ETHICS uses the term for a person who analyses the job-satisfaction and efficiency needs of the User/Management and the establishment. The Analyst in the ETHICS Method provides an opportunity for increasing job satisfaction. The Analyst is responsible for examining the preliminary key objectives, Functions and Designing the technical part of the System. In SASD the SD uses the term as "an informal term for a person whose job is to analyse the user's need, and to then derive the functional requirements of a system" [P-423, 20]. The SA uses the term without defining it explicitly. It is used for a middleman between the User community and the programming community.

Note:

1. In the Life Cycle the term System Analyst is not used purposely, because the analysis tasks in the Life Cycle may start from a point where there is no System. In a case where there is no System what will the System Analyst analyse?

2. The term System Analyst is used differently by these Methods/Methodologies and each has assigned different scope and duties.

3. The ETHICS Method mixes the duties of the System Analyst and the System Designer. Both the terms are used without any discernable difference.

Channel:

JSD defines the term Channel as "an

implementation device by which a process is inverted with respect to two or more data streams simultaneously" [P-374, 21]. MASCOT defines and uses the same term as: "An inter-communication data area used exclusively for passing message data between activities" [22]. The term is not used by LSDM, ETHICS, SADT and SASD.

Note: The same word is used to define different purposes and concepts.

<u>**Control:**</u>

The word control is not used by JSD & LSDM as a term.

MASCOT uses another term Control Queue which is a data structure providing the focal point for Software stimulation and mutual exclusion of one activity by another. It is a basic MASCOT variable. In SADT the term control is used but not defined explicitly. It is used as a constraint or a controlling factor. The control governs how a box should perform its role. It is used to show the control relationship among the boxes. It is represented by an arrow and enters from the top of the box. The output of a box may provide some or all controls for one or more other boxes.

In SASD the SD uses the word control or control of information frequently without defining it explicitly. The SD uses the word control as a piece of information which passes among the Modules. In SASD Control usually subordinates the Modules possesed by its superior But it is not necessary, and its inverse is also possible, though not desirable. It is imperative information having a strong verb, e.g. READ NEXT RECORD, CANCEL ORDER, STOP WRITING etc..

Note: It is not clear how to determine or fix a control in SADT. Further the SADT control does not represent the flow of control or sequence. In SASD the imperative verbs are used to represent both Control and Action. It is not clear how to make the distinction between the SASD Control and Action. In SD if two Modules communicate by passing a piece of information that can control the logic of the other Module, such Modules are called control coupled Modules. Control coupling covers all forms of connections that communicate elements of control e.g. actual transfer of control, passing of data that changes, regulates or synchronizes the target Module. This type of coupling is non-essential. A System that includes control coupling means that its Modules are less independent and subordinate Modules are not black boxes. In SADT control does not represent the flow of control or sequence.

Data Structure Diagram:

JSD defines the term as "A Structure diagram representing the ordering, of records in a data stream or of successive state vectors" [21]. LSDM uses the term Entity Model for some thing similar to the JSD Data Structure Diagram. The LSDM Entity Model represents the generic data structure of the System's data and that data is used for System Development. MASCOT and ETHICS do not use the term Data Structure Diagram. SADT uses the term datagram for a similar purpose as the JSD Data Structure Diagram and LSDM Entity Model. The SADT datagram represents the data aspects of SADT model in terms of things. SD uses the term Data Access Diagram similar to the JSD Data Structure Diagram. The term Data Access Diagram is defined as "A graphic tool for depicting the ways by which a data store can be referred to by means of the information contained in another data store" [61].

Data Store:

JSD uses two different terms i.e. Data Stream and State Vector as a near synonym to a store of data, data element and data structure. The JSD term Data Stream is defined as "an ordered set of records or messages by which two processes communicate, the records being read by one process in the order in which they were written by the other". The term State Vector is used as "local variables of a process, including text pointer" [21]. LSDM uses the term Data Store without defining it explicitly. Data Store in LSDM is concerned with organising the operational System such as transactional files, extracted files for sorting & listing, etc.. Each entity in the LSDM Entity Model is recognised as a Data Store or part of a Data Store in the DFD. MASCOT uses the terms IDA, Channel & Pool for the purpose of storing the data. The term IDA is defined as a "generic term for the data areas which provide the exclusive means by which activities communicate namely channels & pools" [22]. The term channel is used exclusively for passing message data between activities.

The access to a channel is essentialy made via access procedures. Channel has two directional interfaces one for sending and the other for receiving. The term pool is defined as "an inter-communication data area used to form reference data which can be accessed by one or more activities. Access to a pool is by user defined conventions but can be via access procedures" [22]. ETHICS and SADT do not use the concept of Data Store. In SASD the SA uses the term without defining it explicitly as a place where data, data elements and data structures are held from one process to the next until needed or kept permanently. The data in the data store enters via data flow and cannot exit unless it has been stored.

Note: The above terms are defined and used differently for similar things and for similar purposes. The term data store and data stream, state vector, pool and channel represent the similar things or their purpose is more or less the same.

Design:

The basic term Design is defined in the Life Cycle as a verb and a noun. As a verb it is defined as conceiving & planning by using an agreed proposal(s) and its Specifications as input and converting the plan into an unambiguous executable model, called a System frame or System skeleton to perform its purpose(s) i.e. Function(s). As a noun it is defined as a tool set which accepts the human proposal(s) and its Specifications as input and can develop an unambiguous executable System frame or System skeleton to provide the information to the User/Management and serve his purpose(s) i.e. Function(s).

The term Design is not defined by JSD, LSDM, MASCOT, ETHICS and SADT. The SD defines the term as "to plan the form and method of a solution" [p.410, 20]. It is also defined as "......... the process of transforming what has to be done into a means of doing it" [p.1.2, 18]. SA defines the term as: "The (iterative) process of taking a logical model of a system together with a strongly stated set of objectives for that system and producing the specification of a physical system that will meet those objectives" [P-176, 12].

Note:

1. The basic term Design, its scope and concept is not defined uniquely by the Methods/Methodologies.
2. The 'Process' is not used by SA in the way it has been defined by SD.

<u>**Designer:**</u>

The term Designer is defined in the Life Cycle as: A person(s) who: (1) Uses proposals developed by the Analyst to develop system specifications. (2) Transforms the system specifications into a design to meet the Requirements of the User/Management. (3) Is concerned with designing details of a system, subsystems, modules, etc.. The term Designer may be used as a singular and plural and for male and female.

The term Designer is not used by JSD, LSDM, and

ETHICS. However, JSD and LSDM use a new term Developer without defining it, to mean some one who develops a System. The MASCOT, SADT and SASD use the term Designer without defining it. In SASD the SD uses the term for a person who is concerned with structural Design of a program or System.

Entity:

The term is defined by JSD as "an object in the real world which participates in a time ordered set of actions" [21]. LSDM uses the term Entity frequently without defining it explicitly. Each entity in the LSDM Entity Model is recognised as a data store or part of a data store in DFD. MASCOT and ETHICS do not use the term Entity. SADT uses the word Entity in its general sense i.e. It is used for things or objects such as men, machines, company, etc.. SA uses it for a source or destination of data on DFD. SA also uses the term for something about which information is stored in a data store e.g. men, machine, workers, etc..

Note: There is no clear and distinct criteria to select the Entities which must be in the System. The selection depends on the personal choice of the Analyst/Designer which may be different if selected by other Analyst/Designer.

Event:

The term Event is used by JSD & LSDM without defining it. JSD uses the word event for the result of action & participation of one or more entities in the real world. The Events in LSDM modify or update the System's data in the System and they affect the System. MASCOT, ETHICS, SADT, and SASD do not use the term Event.

Note: The same word is used by JSD and LSDM for different purposes.

Implementation:

The term is used in the Life Cycle for the tasks which are concerned with giving practical effect to the Design or System skeleton; the manipulation & translation into code of the Software System frame developed earlier in the Design Phase, to fit it to the Hardware and in some cases to the Software also. It is also concerned with debugging the code, if necessary.

The term Implementation is used but not defined explicitly by JSD & LSDM. JSD uses the term for the jobs in which the Specifications are transformed so that they may conveniently and efficiently be executed. LSDM uses the term Implementation for the work concerned with the development of operating procedures, manuals, training the User/Management and organising the administrative structure of the establishment. MASCOT uses the term without defining it. It is used by MASCOT to implement the Software on host & target machines. ETHICS uses the term for a set of change processes in which an existing System is assisted to move smoothly and successfully from one organisational state to another. SADT has also not defined the term Implementation and is not concerned with the details about Implementation tasks. SD defines the term as "The activity of a project during which the design of a system is tested, debugged, and made operational" [61].

Note:

1. The Implementation tasks covered by the Methods/Methodologies are limited when compared with those of the Life Cycle. For example the Implementation activities covered by the SD does not encapsule how to fit the System into Hardware or to other existing Software Systems, to prepare operating procedures, manuals, training of the User/Mamagement etc.. It also does not say how to schedule the Modules, to check whether the Hardware and Software (in some cases) are available and are in working condition etc..
2. The scope of Implementation tasks covered by different Methods/Methodologies vary too much.
3. The scope of Implementation tasks in the ETHICS Method is mainly concerned with the Transition tasks.
4. The Implementation activities in SD start after drawing final structure charts. These do not overlap with any other activities to save time. Whereas in the Life Cycle the Implementation activities

may overlap with other Phases of the Life Cycle.

Logical:

The term Logical is not used by JSD, MASCOT, ETHICS and SADT. LSDM uses the term too frequently without defining it explicitly. Perhaps the term is used for something which is not Physical. LSDM belives that "the term logical is a euphemism for ambiguous, general, vague or meaningless" [p.8, 16]. But the terms Logic, Logical Data Structure, Logical Frame-work, System Logic etc. have been frequently used without defining them explicitly. SASD defines the term logical as a "non-Physical (of entity, statement, or chart): capable of being implemented in more than one way, expressing the underlying nature of the System reffered to" [P-235, 12].

Note: It is observed that Logical is one of the terms which is misused and is used differently by the Software Engineers.

Management:

The Life Cycle defines the term as: (1) The collective body who handles and conducts the running and operations of an organisation for which the system is to be developed. (2) The authority or controlling body whose Requirements are to be evaluated, understood and the results of the study to be submitted to them. (3) One who directs and handles the System Development team. (4) The chief or director of a data processing section or computer section within an organisation.

The JSD, LSDM, MASCOT, ETHICS, SADT and SASD have used but not defined the basic term.

Method/Methodology:

The term Method in the Life Cycle is defined as a way, set of Phases or steps which should be followed in order to perform certain System development tasks, such as Project Management, Analysis, Design, Implementation, etc.. It is designed to do certain specific tasks of System Development and is available for small or specific types of projects. It may be considerd as a subset of a Methodology.

The term Methodology in the life Cycle involves philosophies, Management techniques, Methods, Phases or steps, rules, techniques, tools or aids and documentation required to develop a System. In principle it is concerned with the complete span of System Development tasks. A Methodology may contain more than one Method or Phases, etc., to do different System Development tasks together with the rules to apply them.

JSD has not defined the term Method or Methodology explicitly, though it is used. JSD uses the terms as: "A Method is a way of doing some thing; methodology is, or should be, the study and science of method" [p-368, 21]. LSDM, MASCOT, ETHICS and SASD have not defined what they mean by the basic terms Method or Methodology. MASCOT uses the term inconsistently. For example, at [22], [P-1, 74], [P-1, 75], [P-6, 87], [88].

MASCOT uses sometimes Method and sometimes Methodology. In fact MASCOT like others is not clear about the terms Method and Methodology. However, SADT uses the term Methodology, without defining it explicitly, for a coherent, integrated set of Methods and rules that form a disciplined approach to analyse & design. SADT uses the term Methodology consistently.

Note: It reveals that the authors of the above Methods/Methodology have no clear and agreed concept of the terms. Further JSD & SADT have a limited concept and scope of the terms.

Process:

The JSD defines the term as: "Execution of a program text from beginning to end, or a particular instance of such execution" [P-376, 21]. JSD classifies the term Process as Model Process and Function Process. The term Function Process is defined as: "A process which is added to the system for the purpose of producing output, as opposed to model process" [21]. The JSD Process is a statement of time ordering of events whose text is similar to a structured program which contains Specifications of the JSD Model and Functions. The term Function Process is perhaps equivalent to the term Module used by SD. LSDM, ETHICS and SADT use the term without having any definition. Perhaps it is used in its general sense. MASCOT uses the term Activity for Process. In SASD the SA defines the term Process as: "A set of operations transforming data, logically or Physically, according to some process logic" [12]. The SD defines the term Process as "An activity in data flow diagram that transforms input data flow(s) into output data flow(s)" [p-346, 61]. In SASD the SD Process takes data and processes it i.e. it transforms data. SD uses a limited scope of Process e.g. an SD process cannot be activated or performed without any data. A Process in general may or may not perform a Function. But in SD a Process which performs a Function is called a Module.

Note:

a. Maddison et al., define the term Process during the study on Information System Methodologies as "Sequence of actions, operations, changes; e.g. using and creating data" [p.109, 65].

b. The steering committee of the International Workshop on the Software Process and Software Environment defines the term Software Process as "The collection of related activities, seen as a coherent process subject to reasoning, involved in the production of a software system" [66].

c. Again the word process is one of those words which are used commonly by the Methods/Methogologies but is either not defined or definition varies.

<u>Requirements Specification:</u>

The term in the Life Cycle is used as: A Requirements Specification is a particular and precise statement of those things which are deemed to be needed. This perception of needs is from the view of the User, their Management and the environment.

JSD, MASCOT and ETHICS do not use the term Requirements Specification. LSDM uses a new term called PRL i.e. Problem/Requirement List. This list describes formally the existing System's problems and the new System's Requirements. This term which is used for the list of problems/requirements may to some extent be close to the term Requirements Specification. In SASD the term 'Functional Requirement' is defined by SD as: "A precise description of Requirements of computer System; includes a statement of the inputs to be supplied by the user, the outputs desired by the user, the algorithms involved in any computations desired by the user, and a description of such physical constructs as response time, volumes, and so on" [20]. The above definition of Functional Requirements is very close to, and is a particular case of the term Requirements Specification used in the Life Cycle.

System:

The term System in the Life Cycle is used: (1) To represent a set of interconnected items or elements arranged in a certain order, e.g. Management of an organisation, a computer System, a railway network, etc.. (2) For the representation of the Design model or Design frame for an environment. (3) In the domain of computer science, any combination of Hardware, Software and the User/Management to fullfil the specific Function(s) or purpose(s) in an environment.

The basic term System is not defined by JSD, LSDM, MASCOT, ETHICS, SADT and SASD. However, the MASCOT System essentially has an environment containing actuators & sensors, people and software. The people & software control and monitor the actuators & sensors. SADT uses the term for any combination of Hardware, Software, people or may consist only of people [p-2.2, 11].

Note: The most basic term is not defined and used clearly and uniquely by the Methods/Methodologies.

System Specifications:

The term System Specifications is defined in the Life Cycle as a precise statement of the Requirements in a form which is meaningful to the Designer. This statement is stated in the technical terms of computing and is considered as an interface between the Analyst and Designer. To develop the System Specifications the Life Cycle requires a complete Phase called System Specifications Phase. This Phase is an interface between the Analysis and the Design Phases. This Phase separates completly the Design work from the Analysis and refers to the information which must be delivered from the Analysis to the Design Phase. This Phase converts the detailed proposal into clear and unambiguous statements stating what the System must do.

JSD uses a general term i.e. 'Specification' for the term System Specifications without defining it formally. By the term Specification they mean a set of Process which can reflect the behaviour of the entities, can create the outputs, and the information about the System timings. These are used to specify the real world model and show the Functions required by the User/Management with respect to that model. SADT does not use the term System Specifications but uses another term called Functional Specifications for the similar purpose without defining it explicitly. The Functional Specifications in SADT are used as an input to the System Design Phase. In SASD the SD uses the term System Specification as a synonym for Functional Requirements [P-413, 20] and the Functional Specification as a synonym for Functional Requirements [P-413,20]. In fact the terms System Specification, Functional Requirements and Functional Specification used by SD are a synonym for each other and are used for the same things. Also another term the Logical Functional Specification is expressed in SA as a detailed statement of what the system is to do, which is as free as possible from the physical considerations of how it will be implemented [P-24, 12]. Further more a term 'Structured Specification' is defined in SD as "a structured model of manual and automated procedures of a system; specifically, the target document of structured analysis comprising data flow diagrams, a data dictionary, mini-specifications, data access diagrams, and a minimal amount of additional information" [61].

Note: It shows that the above terms are used for similar purposes by the Methods/Methodologies and more interestingly by the same Methodology at different points. It further reveals that these Methods/Methodologies are careless in using and defining the terms. Perhaps these Methods/Methodologies relate, as a fashion or trend, everything with the terms Function, Functional and Functionality. The terms which are themselves not clear, are used and defined ambiguously by the Methods/Methodologies.

User:

The term User is defined by the Life Cycle as: (1) As a person who will be using the system such as an operator, software programmer, quality controller, stock control officer, accountant, etc.. (2) As a person who will be using the system in a supervisory capacity such as a person who is in charge of section in a bank, quality control supervisor, etc.. (3) As a person or organisation who will be using the system as the owner, for example, a government ministry, a banking organisation, etc.. (4) AS a person who will be using the system without knowing anything about the Hardware or Software, but is supposed to know clearly about the results he is expecting and be able to check the input or output, e.g., bank customer, user of flight information services, etc.. (5) As the clerical and other people who will work directly with the system by using terminals, filling output forms or interpreting output for their jobs etc.. (6) As a person or organisation whose requirements are to be evaluated and understood and the result of the study is then submitted to them for further actions, e.g., bank authorities, government officials, etc.. (7) As anyone who uses the system by issuing commands. (8) But it excludes Analysts and Designers and anyone who supplies the system.

The term User may be used as singular and plural and for male and female.

JSD, LSDM, MASCOT, ETHICS, SADT have used the term without defining it. In SASD the SD defines the term as: "An informal term describing the person, persons, or organisation that expects to benefit from the development of a computer system" [P-425, 20]. SA does not define the term.

B.3 ETHICS

34. What are the key words used by ETHICS Method?

The following are the important key words which are frequently used by ETHICS Method.

Analyst, Discrepancy Analysis, Implementation, Job-satisfaction, Participative Approach, Participative System Design, Process, Socio-technical Approach, Socio-technical System, Social Structure, System Analysis, Technical System, Unit Operation, Variance, Variance Matrix.

35. Which of the above key words are not used according to their definition given by IEEE?

Analyst:

The term Analyst or System Analyst is not defined by IEEE.

The term is used but not defined by ETHICS Method. It is used for a person or a group of persons who analyses the job-satisfaction and efficency needs of the User/Management and the establishment. The Analyst provides an opportunity for increasing job-satisfaction through the redesign of work and considers different aspects of the new System. The Analyst is responsible for examining the preliminary key objectives and functions and designing the technical part of the System.

Note: Some times a common term System Designer is used for the terms Analyst and Designer with out defining it explicitly. The term function is used but not defined. Also it is not mentioned how a function is determined and represented.

Discrepancy Analysis:

The term is not defined by IEEE.

The term is used by ETHICS Method but not defined. It is used to cover an identification of variance & examination of existing task structure and work-flows. This is done to establish the extent to which there is a good or bad fit between what exists, what should be, and what has been achieved.

Implementation:

IEEE defines the term as: (1) "A realization of an abstraction in more concrete terms; in particular, in terms of hardware, software, or both". (2) "A machine executable form of a program, or a form of a program that can be translated automatically to machine executable form". (3) "A process of translating a design into code and debugging the code" [34].

The term is used by ETHICS Method for a set of change processes in which an existing System is assisted to move smoothly and successfuly from one organisational state to another [77]. The term involves providing the user with a commitment to the new work System, with an understanding of how it functions and with the skills to operate it efficently [p-97, 8].

Note:

a. The scope of both definitions is different. The definition used by ETHICS Method is more concerned with the Transition activities.

b. In the above definition the word process is used by ETHICS Method in its common use i.e. it is not used in its technical sense.

Job-satisfaction:

The term is not defined by IEEE.

The term is defined by ETHICS Method as: "The 'fit' between what an individual or group is seeking from the work situation and what they are receiving from it, in other words the 'fit' between job needs and positive expectations and the requirements of the job" [p-37, 8]. The 'fit' is also defined as "achievement of a good fit between job needs and expectations and job experience" [p-15, 9].

Note: Job satisfacion can be met when personality needs, competence & efficiency needs and needs associated with personal values are fulfilled.

Participative Approach:

The term is not defined by IEEE.

The term is defined by ETHICS Method as: "A participative approach to work design means that the employees of a department or their representatives construct a new form of work organisation which is based on a diagnosis by them of their own needs" [p-7, 8].

Participative System Design:

The term is not defined by IEEE.

The term is defined by ETHICS Method as: "Handling responsibility for the Design of a new work System to the employees who eventually will have to operate it" [p-5, 77]. The term is used for a Design which is based on the participative approach. There may be three types of participative Design, i.e. Consultative Design, Representative Design and Consensus Design.

Process:

IEEE defines this term as: (1) "In a computer system, a unique finite course of events defined by its purpose or by its effect, achieved under given conditions". (2) "To perform operations on data in process" [34].

The term Process is used by ETHICS Method but not defined. The term is used in its general sense i.e. is not the same as that used by JSD, IEEE, etc..

Socio-technical Approach:

The term is not defined by IEEE.

The term is defined by ETHICS Method as: "A design philosophy that produces productivity, quality, co-ordination and control; but also provides a work environment and task structure in which people can achieve personal development and satisfaction" [p-6, 77]. It is used for an approach which recognises the organisations as purposive entities. It is an approach which incorporates a systematic Analysis of the technical components of the existing System and the grouping of those into unit operations. This is an approach for improving the efficency of an existing System through identifying and Analysing System variances.

Socio-technical System:

The term is not defined by IEEE.

The term is defined by ETHICS Method as: "A socio-technical System is any unit in the organisation composed of a technological and a social subsystem having a common task or goal to accomplish" [p-14, 77].

Social Structure:

The term is not defined by IEEE.

The term is defined by ETHICS Method as: "The network of roles, relationships and tasks which interact with the technical System" [p-14, 77].

System Analysis:

The term is not defined by IEEE.

The term is not defined by ETHICS Method. However, It is concerned with the preliminary examination of key objectives and functions. It involves the following:

a. Specifying in detail the nature of the problem or opportunity. Why should an existing System or Systems be re-designed?

b. Identifying those groups with an interest in how the problem is solved.

c. Defining the boundaries of the System with which Design is concerned.

d. Specifying the preliminary objectives of the System.

e. Identifying the principal unit of operations i.e. sub-systems.

Note: The term System Analysis which is most frequently used in the field of computer science is not defined by ETHICS or IEEE.

Technical System:

The term is not defined by IEEE.

The term is defined by ETHICS Method as: "A system which consists of tools, techniques and procedurs used for processing the raw materials of information" [p-14, 77].

Unit Operation:

The term is not defined by IEEE. However, the closest possible term for Unit Operation may be Sub-system which is defined as: "A group of assemblies or components or both combined to perform a single function" [34].

The term is defined by ETHICS Method as: "A set of integrated tasks separated from the other sets of tasks by some change of state in the input" [p-161, 8]. It is also defined as: "An integrated set of activities which are attached to a major System function. This set of activities is separated from other sets of activities by some kind of boundary _ a change of input, time, location etc." [p-8, 77].

<u>Variance:</u>

The term is not defined by IEEE.

The term variance is defined by ETHICS Method as: "A tendency for a work System to deviate from a desired specification" [p-8, 8]. The term is used for a weak area in a System; a part of the System which tends to deviate from some desired standard or norm.

Note:

a. The term specification is used but not defined, neither is it mentioned how and from where to obtain or develop the specification.

b. Some variances may be called the key variance. These are the variances which are most direct or important in their impact or effect.

<u>Variance Matrix:</u>

The term is not defined by IEEE.

The term is used but not defined by ETHICS Method. It is a matrix where variances are placed. The Matrix shows the influences of a problem i.e. how it interacts with other problems in the unit operation. In this matrix the variances which are associated with the System input are placed first then those which are associated with the System output. The interaction a variance has with other variances is noted by placing its number in the vertical column it heads opposite those variables which it effects. This matrix can easily point out those variances which affect more and creates many other variances.

Note: It is a unique approach which no one has used during the System Analysis.

B.4 JSD

34. What are the key words used by JSD?

The following are the important key words which are frequently used by JSD.

Action, Data Base, Data Stream, Entity, Event, Function, Model, Process, Processor, Real World, Specification, State Vector, System.

35. Which of the above key words are not used according to their definition given by IEEE?

Action:

IEEE does not define this term.

JSD defines the term Action as "an event in which one or more entities participate by performing or suffering the action".

Note: In the above definition the word event is used by JSD without defining it. Further the word action is used for defining the word Action itself. These make the definition ambiguous. However in their literature it is clear that Actions in JSD take place in the real world outside the System. The Actions take place at a point in time not over a period of time. These Actions are essentially atomic and can not be decomposed further in sub actions.

Database:

IEEE defines this term as: (1) "A set of data, part or the whole of another set of data, and consisting of at least one file that is sufficient for a given purpose or for a given data processing system". (2) "A collection of data fundamental to a system". (3) "A collection of data fundamental to an enterprise".

This term is not formally defined by JSD. But they use it as a collection of state vectors of System processes: model process, function process, and some times schedular process.

Note: Database development in JSD is done at the Implementation step, at the end of development activities. In the Implementation step the Designer may keep the state vector of one process in one, two, or many Database segments or records. A JSD Database record can easily accommodate the state vectors of two or more processes. The concept of Database differs from that of the IEEE in the sense that it is used in a specific sense and in terms of the jargon of the JSD terminologies.

Data Stream:

IEEE does not define this term.

JSD defines the term Data Stream as "an ordered set of records or messages by which two processes communicate, the records being read by one process in the order in which they were written by the other".

Data stream is considered as an infinite expansable queue which consist of a sequence of records. These records can

be read or written. Data streams are used by the processes for the purpose of communication. They are also used to connect two processes and the System to the real world.
Note: To a certain extent this term is used in a similar way to the data flow concept in SASD Methodology.

Entity:

IEEE does not define this term.

JSD defines the term Entity as "an object in the real world which participates in a time ordered set of actions".
Note: The meaning of JSD Entities is closely associated with the process model and with the time ordered set of actions. JSD Entites are objects which describe the real world e g. customer, elevator, ship, book etc.. The JSD Entities perform or suffer an action, which have significant time ordering. Each separate active and independent Entity in JSD is represented by processes. The criteria for selection of an Entity is that the description must be able to support the functions to be provided by the System and must be able to do an action. Therefore an Entity cannot be omitted if its absence will affect a function. This concept of Entity differs from that used in database Entity. So a JSD model has less entities than a database model.

Event:

The term is not defined by IEEE and JSD, though it is used very frequently.

Function:

IEEE defines the term Function as: (1) "A specific purpose of an entity or its characteristic action". (2) "A sub program that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with sub-routine".

JSD has not define this term explicitly. The Method uses the term as an action or set of actions performed by the System and resulting in the production of output.

Note:

1. It is claimed that a model of the real world can provide a conceptual as well as computational basis on which various Functions of a System can be built.
2. JSD claims that from the set of words called dictionary which is expressed in JSD model the verbs can be used to specify the Functions. It is not clear how to select and determine the verbs to be used as Functions. Further the meaning of the words depends upon the User's/Management's view of reality.

Model:

IEEE does not define this term.

JSD defines the term Model as "An abstraction which has been realized, especially by a set of sequential process".

JSD uses this word to mean a Model of the real world outside the computer System, which is to be developed. It is not used as a model of the System itself nor its attributes or characteristics. Firstly it develops an abstract description of the real world which is simply a description in which relevant aspects are described and irrelevant aspects omitted. Then secondly it makes a realization in the computer, of that abstract description. The JSD Model provides the conceptual and computational basis for the System functions. It concentrates on establishing the basis on which a set of functions can be specified. It does not contain any description of the System output.

Process:

IEEE defines this term as (1) "In a computer system, a unique finite course of events defined by its purpose or by its effect, achieved under given conditions". (2) "To perform operations on data in process".

JSD defines Process as "execution of a program text from beginning to end, or a particular instance of such an execution".

The JSD Processes are statements of time ordering of events. Its form is like a simple structured program. The Process text contains both the specifications of the model and the specifications of the function. It forms a part of the System Specifications. One Process is used for each independent active entity, which may have local variables to represent attributes of the entity. A Process may have any number of input/output data streams. The connection between the JSD Process is performed by state vector connection or data stream connection. Each Process may be considered to belong to a Process class eg. class of customer Process, class of order Process etc.. The class of Process determines its program text and also determines how the Process fits into the System network. In a Process only one thing can happen at a time. Therefore there must be several Processes in a JSD model. To make the model dynamic JSD has developed a network of such Processes.

Note: JSD is close to the first definition of IEEE, with a special condition of time ordering of events.

Processor:

IEEE does not define this term.

JSD defines the term Processor as "a device used in implementation, capable of realizing a single process execution".

Real World:

IEEE does not define this term.

JSD defines the term Real World as "the context in which the subject matter of the System exists".

Note: Their literature reveals that the JSD Real World is the starting point of the Methodology. It is the context in which the subject matter of the System exists. It is concerned with the world in which the time dimension has prime importance, and such a world is dynamic in nature. The Real World is described in terms of entities, actions which they perform or suffer, and the ordering of those actions. These entities have fixed properties and fixed relationships.

Specification:

IEEE defines the term Specification as: (1) "A document that prescribes, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or system component". (2) "The process of developing a specification". (3) "A concise statement of a set of requirements to be satisfied by a product, a material or process indicating, whenever appropriate, the procedure by means of which it may be determined whether the requirements given are satisfied".

JSD has not defined the term formally, though they are using it frequently. By Specification JSD mean the set of processes which can reflect the behaviour of the entities, can create the outputs and information about the System timings. JSD Specifications are based on actions, their attributes and time orderings. These are used to specify the User's/Management's model of the real world and also the functions required by them with reference to the model. JSD develops its Specifications which are independent of any software and hardware, so these Specifications can be executed directly on a machine. The JSD Specifications are created from the information about the real world provided by the

User/Management. JSD Specifications are written as a set of communicating, concurrent network of processes. Their concept of Specification is very close to that of IEEE. However they differ in the sense that the Specifications developed by JSD specify the User's/Management's model of the real wold and also their required functions; where as IEEE specifies Requirements, Design, behaviour etc..

State Vector:

IEEE does not define this term.

JSD defines the term State Vector as "the local variables of a process, including the process text pointer".

Note: It is a form of connection between the two processes, and also the real world and the model process. It changes only when a process is executed. When State Vector connections are used between the two process, one of them directly inspects the State Vector of the other.

System:

IEEE defines this term as: (1) "A collection of people, machines, and methods organised to accomplish a set of specific functions". (2) "An integrated whole that is composed of diverse, interacting, specialised structures and sub-functions". (3) "A group or sub-system united by some inter-action or interdependence, performing many duties but functioning as a single unit".

JSD does not define the term System explicitly.

Note: From their published material it is clear that they consider the System as the simulation of the relevant parts of the real world, which are outside the System. The subject matter of the System is obtained from the real world so that the System can reflect the real world as it is. In JSD System functions which are required by the User/Management are introduced into the System at a later stage. A JSD System has definite boundaries, across which the input/output between the real world and the System flows. It consists of computer and manual procedures and hardware. JSD uses the term System in a restricted way. JSD considers it as a simulated model of the relevant part of the real world, where as IEEE uses it as the collection of relevant parts to do a set of functions.

B.5 LSDM

34. What are the key words used by LSDM.

The following are the important key words which are frequently used by LSDM.

Composit Logical Data Design, Data Driven System, Data organisation, Data Store, Enquiry Function Catalogue, Enquiry Process Outlines, Entity, Entity Model, Entity Life Histories Model, Event, Function, Logical Data Structure, Logical Process, Logical Specification, Low level Information Process, Physical Document Flow, Physical Design, Problem Requirement List, Process, Process Outlines, Relational Data Analysis, Update Process outlines, State, System Logic, System Requirement Solution.

35. Which of the above key words are not used according to their definition given by IEEE.

Data Driven System:

IEEE does not define this term.

LSDM uses the term Data Driven System without defining it explicitly. The term is used for a System in which generic data structure is used. This generic data structure generally tends to change very little over a period of time.

Data Flow Diagram:

IEEE defines the term DFD as "A graphic representation of a system, showing data sources, data sinks, storage, and processes performed on data as nodes, and logical flow of data as link between the nodes" [34].

The term DFD is defined by LSDM as "Data Flow Diagrams are operational picture of how data moves between the system and the external world".

The LSDM DFDs are used to represent the existing System to find out its information flow, the needs of the User/Management and to define in pictorial form the required System. These diagrams also show how the process, transactions, System-maintainance data fit together.

Note: Apart from the other uses of LSDM DFD, it is also used to find the needs of the User/Management in the existing System.

Entity:

IEEE has not defined the term Entity.

LSDM uses this term frequently, but it is not defined explicitly. However, from their literature it is revealed that they are using the term Entity for the object which actually occurs in the System. Each Entity in the Entity model is recognised as a data store or part of a data store in DFD.

Entity Life Histories Model:

IEEE has not defined this term.

LSDM uses the term as "Entity Life History diagrams provide a third view of data, in order to show the effects of time on the system"

The ELH model shows how an entity changes, in the System over a period of time and in what order the events must occur. A chart is developed for each entity taking part in the System. This chart shows what events the System needs to deal with and which events effect the data in the System. During the drawing of these charts the origin of each event and processing of each transaction is identified. Some other details about error dectection and dealing with unexpected events are also added.

Entity Model:

IEEE has not defined this term.

The term Entity Model is defined by LSDM as "an entity model is the generic data structure of the system's data that is used as a basis for developing the System to support its processing needs".

This model shows the data as the basis for an information System and shows how it can be accessed.

Note: These models are created at different stages of the System development i.e. from the Initial Investigation stage to the detailed Composit Logical Data Design showing data structure, data fields, access paths etc.. The data stores which are concerned with organising the operational System such as a transaction files, extracted files for sorting & listing etc. are not shown in the entity model.

Event:

IEEE has not defined this term.

LSDM uses this term frequently with out defining it explicitly. They are using the term for the effect of actions on entities, resulting in a happening for something in the environment or System. For example an event may create changes to the System data.

Function:

IEEE defines the term Function as: (1) "A specific purpose of an entity or its characteristic action" (2) "A sub program that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with sub-routine".

The term Function is used by LSDM but not defined. It does not say how to find and represent the functions.

Process:

IEEE defines this term as (1) "In a computer system, a unique finite course of events defined by its purpose or by its effect, achieved under given conditions". (2) To perform operations on data in process".

The term Process is used by LSDM but not defined. They use the term in its general sense.

__System:__

IEEE defines this term as (1) "a collection of people, machines, and methods organised to accomplish a set of specific functions". (2) "an integrated whole that is composed of diverse, interacting, specialised structures and sub-functions. (3) "a group or sub-system united by some inter-action or interdependence, performing many duties but functioning as a single unit".

The term System is not defined explicitly by LSDM.

Note: The terms Data Design, Data Analysis, Logical Data Structure, Logical Process, Logical Specification, Low Level Information Process, Relational Data Analysis, System Logic etc., have been used frequently without being defined.

B.6 MASCOT

34. What are the key words used by MASCOT?

The following are the important key words which are frequently used by MASCOT.

Access Procedure, ACP Diagram, Activity, Channel, Construction, Control Queue, Form, IDAs, Implementation, Interrupt Handler, Kernel, Modules, Scheduling, Slice, Stim, Sub-system, System, System Construction, System Element.

35. Which of the above key words are not used according to their definition given by IEEE?

Access Procedure:

IEEE does not define the term Access Procedure.

MASCOT defines the term Access Procedure as "a procedure used to access a channel or pool data structure. Strict conventions are specified for channel access procedures" [22]. It is used as a means by which activities can communicate asynchronously via the IDAs. The Access Procedure provides an interface between the activities and the IDAs i.e. channels & pools. The Access Procedures may be of two types one each for inputs/outputs. There is a complete symmetry between these two types of Access Procedures. The Access Procedures are developed to do the following actions in a specified sequence.

* **Join**-the control queue,
* **Check**-the state of the data areas and wait if needed,
* **Transfer** Data,
* **Amend** State,
* **Stim**-the control queue controlling the other interface,
* **Leave**-the control queue controlling the other interface,
* **Wait**,.

<u>**Activity Channel Pool (i.e. ACP) Diagram:**</u>

IEEE does not define this term. The nearest term for ACP Diagram which is used by IEEE may be the Data Flow Diagram.

MASCOT uses the term ACP Diagram frequently with out defining it explicitly. It is a diagram which depicts activities and IDAs and forms the basis for the division of tasks between various Programmers. The ACP Diagrams can be used to assess the progress in construction, coding and testing on a module to module basis.

Note: MASCOT is heavily based on ACP Diagrams, therefore, developing these diagrams needs highly skilled and experienced Programmers.

Activities:

IEEE does not define this term. The nearest term for Activities which is used by IEEE is process.

MASCOT activity is a process, a piece of program which may be developed independently. It is a unit of DP which is scheduled by a schedular. Activities execute identifiable tasks within a System. Activities are isolated from one another and may be modified independently without having major effects. It is a single thread in the multi-processing System. An activity can have limited communication scope i.e. it is only allowed to communicate with certain data areas. In other words the concept of global data is not allowed. The Activities run in parallel with each other. They communicate via IDAs in an asynchronous way (like JSD scheduling).

Note:

1. The MASCOT Activities are like JSD processes and SASD modules.
2. The manner of communication for Activities i.e. use of IDAs is similar to that of JSD processes which use SV & Data Store.
3. There is no evidence for the claim that the modification of an activity can be done independently without having major effects. It is not clear how the coupling can be minimised to avoid the ripple effects especially when modifications are made.

Channel:

IEEE does not define this term.

MASCOT defines the term Channel as "an inter-communication data area used exclusively for passing message data between activities. All access to a channel must be via access procedures which use basic MASCOT primitives to ensure an orderly and sustained flow of data through the channel" It has two uni-directional interfaces implemented by a pair of access procedures, one for sending the other for receiving. Therefore, a Channel has at least two queues, one for controlling input access and the other for controlling output access. These interfaces are implemented by access procedures. A Channel is represented diagramatically by [----].

Construction:

IEEE does not define this term. The nearest term for Construction used may be IEEE may be Implementation.

MASCOT defines the term Construction as, "all processes necessary to convert source text into executable code resident in a computer" [22]. The process of Construction is used for compilation and loading of individual pieces of software to get the total software content of a computer.

Control Queue:

IEEE does not define this term. The nearest term for this may be data structure. It is a particular case of data structure.

MASCOT defines the term Control Queue as "a data structure, providing the focal point for software stimulation and mutual exclusion of one activity by another. Each control queue has a pending list upon which activities are placed to await either an explicit software stimulus (Stim) or the release of the mutual exclusion (Clear). Other fields within the control queue denote the current state of the control queue and the priority of the current list to which items are transferred from the pending list" [22]. It is a fundemental MASCOT control variable. A Control Queue may have five states of queues. Four primitive procedures i.e. Join, Leave, Wait, Stim are used to interact with Control Queues.

Form:

IEEE does not define this term. The nearest term for Form may be Integration. Integration is "The process of combining software elements, hardware elements, or both into an overall system".

MASCOT defines the term Form as "this operation is used to build MASCOT subsystems from loaded and linked System elements" [10,22]. It is a command used to combine the System elements i.e. channels, pools and root procedures, in order to develop subsystems (and their activities). Before combining the System elements the Form command checks that each item mentioned is a System element and that the parameters which have been given for the root procedure match the Requirements as specified in the root procedure header.

Function:

IEEE defines the term Function as: (1) "A specific purpose of an entity or its characteristic action". (2) "A sub-program that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with sub-routine".

No clear defination of Function is available from the literature of MASCOT Method. Perhaps they are using the term in its general sense.

IDAs:

IEEE does not define this term. The nearest possible term for IDAs used by IEEE may be data base.

MASCOT defines the term as "generic term for the data areas which provide the exclusive means by which activities communicate namely channels and pools", [22].

The Inter-communication data areas are developed to enable the activities to communicate with each other. The IDAs contain the data processed by the activities together with access procedures which perform the tasks like data control, protection & scheduling. IDAs define the external data-interface of an activity. The IDAs can be of two types i.e. 'channels' and 'pools' according to data access requirements. Arrows represents Inter-communication between activities and IDAs.

Note: The use of IDAs has an advantage that it prevents the duplication of work, e.g. a channel developed earlier can be re-used, if needed.

Implementation:

IEEE defines the term Implementation as (1) "A realisation of an abstraction in more concrete terms; in particular, in terms of hardware, software, or both". (2) "A machine executable form of a program, or a form of a program that can be translated automatically to machine executable form". (3) "The process of translating a design into code and debugging the code " [34].

The term Implementation is used by MASCOT but not defined. The term is used to implement the software on host & target machines. The MASCOT Implementation may be of two types i.e. frozen and evolutionary. In the frozen Implementation no change or enhancement in the System is possible, where as in the evolutionary Implementation a set of operational subsystems may be changed while the System is running.

Note: The MASCOT concepts of construction along with Implementation can be equivalent to the IEEE term Implementation.

Interrupt Handler:

IEEE does not define this term.

MASCOT defines the term as "the small piece of machine dependent software responsible for translating a real hardware interrupt into a MASCOT virtual interrupt. Also responsible for remembering and resetting machine status (hardware registers etc.)" [22]. It is a part of kernel software which is concerned with hardware interrupts. It generates a STIM each time a hardware interrupt is received to be applied to a control queue. It interfaces the MASCOT subsystems which are outside the kernel.

Kernel:

IEEE defines the term Kernel as (1) "A nucleus or core, as in the kernel of an operating system". (2) "An encapsulation of an elementary function. Kernel can be combined to form some or all of an operating system or set of firmware". (3) "A model used in computer selection studies to evaluate computer performance".

MASCOT defines the Kernel as "Software necessary to support programming in the form of subsystems. The kernel software includes the despatcher and interrupt handler executive programs, the kernel data base and a set of primitive procedures" [22]. This software provides a set of synchronising primitive operations, two executive routines and a Form command. An important aspect of the Kernel software is its monitor facility which makes possible a time ordered sequence of its actions to be recorded. The synchronisation primitives handle all problems of process synchronisation at the base levels. The two executive routines allocate the processing time. The first one allocates the processor time on demand from external hardware interrupts and the second one is responsible for allocating processor time to the base level i.e. non interrupt driven activities. The Form command is used to combine System elements e.g. channels, pools, root procedures etc.. It also contains certain key files, tables and lists which are used in the construction of software and the dynamic management of executive sequences. The Kernel is adoptable to co-equal multi-processor types of computer. The Kernel makes possible a network of processes i.e. MASCOT activities to be implemented and run on large machines.

Note:

a. The MASCOT definition of Kernel is close to the 2nd. definition given by IEEE.

b. The main point concerned with the MASCOT Kernel is the extent of mutual exclusion needed between the processes. The easiest solution may be to have only a single processor to execute within the Kernel at a time.

c. The Kernel software can not be expressed in terms of a subsystem. It supports the subsystem.

__Module:__

IEEE defines the term Module as (1) "A Program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from, an assembler, compiler, linkage editor, or executive routine". (2) "A logically separable part of a program" [34].

MASCOT defines the term Module as "a basic constructional unit of programming, compilation and loading" [22]. e.g. channel, pool or activity. MASCOT classifies Modules as code Modules and data Modules. Code Modules are mainly instructions to be executed. Its source text is written as a procedure and may contain the name of another Module. Every code Module must consist of at least one procedure. This call may transfer the control to the other Modules at the point of control. Data Modules provide data space for inter-communication between code Modules. Modules can be represented diagramatically either by a circle or by a broken rectangle. The circle represents data processing Modules i.e. activities and the rectangle represents data areas. The Modules after completion are stored in the computer to be linked with other Modules by using MASCOT construction software.

Note:

a. The facility to call other Modules will transfer the control to the other Modules at the point of control. This may create a great problem of coupling and cohesion.

b. The term Module is used for an activity which is defined as process.

c. Again like other authors, MASCOT uses the words Module and process for the same things. These terms are used carelessly and in a confused manner.

Modularity:

IEEE defines the term Modularity as "The extent to which software is composed of discrete components such as that a change to one component has minimal impact on other components" [34].

The term Modularity is used by MASCOT but not defined explicitly. However, it is used for partitioning a software into smaller manageable components. "Modularity starts with partitioning of the system design leading on to the coding of reasonably small modules of source text" [p.1, 22].

Note: IEEE is using the term in the light of Coupling, whereas MASCOT is considering it in terms of its size.

Pool:

IEEE does not define this term. The nearest term for Pool used by IEEE may be data base.

MASCOT defines the term Pool as "an inter-communication data area used to form reference data which can be accessed by one or more activities. Access to a pool is by user defined conventions but can be via access procedures" [22]. The term Pool is used to form reference data which can be accessed by one or more activity. It is basically a depository for non-transit data. In general it is not concerned with data flow through the System. Essentially a Pool has a single bi -directional interface. Diagramatically it is represented as !____!.

Note: It is similar to the terms data store, data base, state vector and data stream.

Root Procedure:

IEEE does not define this term. It is a special case of module.

MASCOT defines the term Root Procedure as "the module which defines the form of the processing for an activity. The formal parameters of the root procedure define the number and type of inter communication areas which the activity will require to access in a running subsystem. Additionally the formal parameters can define the interrupt channels which will be required and they can also be values of any data type" [22]. It is a program code which determines the action of an activity and the function of an entity. It is like a programming unit and may be of any size within practical limits. It has formal parameters which are the channels and pools. These formal parameters of Root Procedure specify the number and type of IDAs which the activity will require to access in a running subsystem. It may be loaded either as a single entity or as a linked collection of several modules.

Scheduling:

IEEE does not define this term.

MASCOT uses the term frequently without defining it explicitly. In MASCOT Scheduling is done by the despatcher, interrupt handler and Scheduling primitives which are themselves parts of the kernel software. The despatcher or interrupt handler calls the activities. The processing can be initiated by the interrupt handler in response to either the hardware interrupts and or by the despatcher in order to perform the pending work listed on the current and delay lists. In both cases control is passed to the activity which in turn is required to co-operate by returning control to the kernel when it is no longer required. Test, Wait, Clear, Stim, Suspend, Delay, Terminate etc. are the examples of Scheduling primitives.

The execution period of an activity i.e. slice is monitored by the MASCOT monitoring facility for measuring and recording slice times. This information is used for establishing satisfactory Scheduling. Despatcher has a privilege that it can detect a very long slice and can forcibly terminate such odd activities. A unified list structure is used for all kernel and control queues list. This enables the despatcher to allocate and adjust the space dynamically to the individual lists without effecting the overall space required. This approach prevents holding up any activity due to lack of list space. **Note:** Its role is like the JSD schedular.

Slice:

IEEE does not define this term. The closest terms used by IEEE for Slice may be Execution Time and Run Time.

MASCOT defines the term Slice as "a period of execution of an activity started either by a hardware interrupt or by the despatcher and terminated by the activity calling a kernel primitive procedure" [22]. The execution of an activity is a period consumed between starting and termination of an activity.

Stim:

IEEE does not define this term. The nearest possible term used by IEEE for Stim may be Instruction.

MASCOT defines the term Stim as "the Primitive procedure which allows an activity to apply a software stimulus to a control queue and hence to a waiting activity (if any)" [22]. It is usually positioned within a special type of channel called an interrupt channel or (INTCHAN). Stim is used as a procedure taking little time to execute.

Subsystem:

IEEE defines the term Subsystem as "a group of assemblies or components or both combined to perform a single function".

MASCOT defines the term Subsystem as "the major unit of software construction in a MASCOT system and the unit which is intoduced into and removed from a running System. A subsystem is formed from System elements and consists of one or more activities" [22]. The MASCOT Subsystem is a network of one or more activities associated with one or more IDAs, for the purpose of convenience. It is functionally related group of modules, used as an allocating unit of creation, starting, stopping and removal. Subsystems are formed from System elements such as root procedures, channels & pools. It shows the structure of the software and its organisation, but it is not an essential part of MASCOT philosphy. Each activity in a Subsystem is essentially a single 'thread' in a multi-processing System. This concept reduces the need of having global System management. The MASCOT Subsystem essentially has named root procedures to support individual activities. The Subsystems are named when they are created. In a situation where such management is unavoidable, the management operations can be done slowly, provided the activities are not effected and global operation can continue normally.

Note:

a. The terms function and functionaly related are not defined. How to identify, specify and represent the functions is not mentioned.

b. From the MASCOT definition it is not clear how many functions a Subsystem can perform. Is it limited to a single function like the IEEE definition of Subsystem?

System:

IEEE defines the term System as (1) "A collection of people, machine, and methods organised to accomplish a set of specific functions". (2) "An integrated whole that is composed of diverse, interacting, specilised structures and sub-functions". (3) A group or subsystem united by some interaction or interdependence, performing many duties but functioning as a single unit" [34].

MASCOT uses the term System frequently without defining it explicitly. It is represented by the current set of running subsystems.

Note:

a. The use of the term System by MASCOT is nearer to the 3rd. definition of System given by the IEEE.

b. The MASCOT System has a characteristic that it can be varied while it is in operation. This can be achieved by adding, starting, stopping or deleting the subsystems.

<u>System Construction:</u>

IEEE does not define the term System Construction. The nearest term used by IEEE for this purpose may be 'Implementation'.

MASCOT uses the term System Construction frequently without defining it explicitly. It is used for the Implementation tasks. The term is used for all those processes necessary to convert source text into executable code in machine. It covers compiling, loading and linking. The basic unit of construction requires several distinct operations supported by various files to record intermediate results of construction processes. These operations are Compile, Load, Link and Form. The subsystems table and activity table, which are the important parts of the kernel data base, are also performed in construction processes.

Note: This tasks is similar to the concept of Implementation of the Life Cycle.

System Elements:

IEEE does not define the term System Element. The nearest term used by IEEE for this purpose may be 'Component' which is defined as "A basic part of a system or program" [34].

MASCOT defines the term System Element as "loaded and linked root procedures, channels and pools which can be used to form MASCOT subsystem" [22]. The root procedures, channels and pools are known collectively as System Elements. These are built up individually by the constrution facilities. These are the component elements which are combined together into subsystems by using the Form facility.

B.7 SADT

34. What are the key words used by SADT?

The following are the important key words which are frequently used by SADT.

Actigram, Activity, Datagram, Design model, Function, Functional Analysis, Functional Model, Functional Specifications, Mechanism, Process, SADT Model, System, System Design.

35. Which of the above key words are not used according to the definition given by IEEE?

Actigram:

The term Actigram is not defined by IEEE.

This is a diagram which represent the activity aspects of the SADT model in terms of their happenings. Actigrams are composed of rectangular boxes and arrows connecting the boxes. Boxes represent activities and their components in the break down structure. The interface between boxes is done by arrows which represent data. The name of the Actigram box starts with a verb representing a function. A box may perform various parts of its activities i.e. functions.

Note: This diagram is concerned with the results of the actions performed by the entities. The result of an action is described by other Methods/Methodogies as an event.

Activity:

The term Activity is not defined by IEEE.

The term Activity is not defined explicitly. Activities are happenings such as CONTROL FLIGHT, MANAGE RESOURCES, CULTIVATE, PRODUCE, ADD ALLOWANCES, etc.. These activities are performed by entities such as man, machines, computers etc.. Perhaps it is used an alias for function.

Control:

The term Control is not defined by IEEE.

The term Control is not defined explicitly. It is used for a constraint or a controlling factor to govern the way needed by a box to perform its role. It is used to show the control relationship among the boxes and is represented by an arrow and enters from the top of a box. The output of a box may provide some or all controls for one or more other boxes.

Note: It does not represent the flow of control or sequence.

Datagram:

The term Datagram is not defined by IEEE.

This is a diagram which represents the data aspects of SADT model in terms of things. Boxes represent the objects, nouns i.e. the data or class of data. The interfaces between these boxes are made by arrows which represent the activities.

Design Model:

The term is not defined by IEEE.

The term is used for a model which can express how the System will be implemented to perform the functions.

Function:

IEEE defines the term Function as: (1) "A specific purpose of an entity or its characteristic action. (2) "A subprogram that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with sub-routine".[34].

The term Function is not defined explicitly by SADT. No clear concept of function is available from their literature. Perhaps they use the term Function to represent an activity i.e. happening which is represented by a verb in a natural language. In SADT Functions are represented by boxes in the Activity diagram. No criteria for detecting activities is given. It entirely depends on the intuition of the Analyst/Designer.

Functional Analysis:

The term Functional Analysis is not defined by the IEEE.

SADT uses the term without defining it explicitly. The nearest synonym for the term Functional Analysis may be the Functional Decomposition, the term defined by IEEE [P-17, 34]. The SADT term Functional

Analysis starts with a statement of System Requirements and delivers Functional Specifications as output [p.i-ii, 11]. During Functional Analysis generally emphasis is made on analysing and documenting what the System should do rather than how it will do it. However, in some cases consideration about how also takes place.

Functional Model:

The term Functional Model is not defined by IEEE.

The term Functional Model is not defined explicitly. However, it is used for a model which can express what the problem is and what functions the System must perform. It expresses activities & data aspects of the System. This model delivers a part of the Functional Specifications.

Functional Specifications:

The term Functional Specification is defined by IEEE as "A specification that defines the functions that a system or a system component must perform" [34].

The term Functional Specifications is not defined explicitly by SADT. However, the term is used to name the output of the Functional Analysis Phase. The major portion of the System Specifications is obtained from the SADT model and serve as one of the inputs to the Design Phase. The major components of the Functional Specifications are provided by the activity diagrams and data diagrams.

Mechanisms:

The term Mechanism is not defined by IEEE.

The term Mechanism is used to indicate 'how' and 'what' is to be realised. It is used to Implement the required functions. A Mechanism may be very abstract or very concrete. If a Mechanism is itself complex then it can be decomposed to a lower level showing how that Mechanism can be carried out by further Mechanisms. The criteria for Mechanism selection is the experience of the Designer and a knowledge about its performance.

Process:

IEEE defines this term as: (1) "In a computer system, a unique finite course of events defined by its purpose or by its effect, achieved under given conditions". (2) "To perform operations on data in process" [34].

The term Process is used but not defined by SADT. They use the term in its general sense.

SADT Model:

The term SADT model is not defined by IEEE.

"An SADT model is a graphic representation of the hierarchic structure of a system, clearly revealing the relationship of all system elements or functions" [p.2-6, 11].

An SADT Model is an organised sequence of diagrams. It mainly consists of actigram, datagram, text in a natural language, glossaries, and node index. The model is developed in a structured manner and can depict the details gradually. Each diagram is connected into the model to express the whole System, taking into consideration their logical relationship with each other. This connection is symbolically made by using the arrows which can only represent constraints and relationships but not the flow of control or sequence.

System:

IEEE defines this term as (1) "a collection of people, machines, and methods organised to accomplish a set of specific functions". (2) "an integrated whole that is composed of diverse, interacting, specialised structures and sub-functions. (3) "a group or sub-system united by some inter-action or interdependence, performing many duties but functioning as a single unit".

SADT defines the term System as "A system may include any combination of hardware, software, people, or may consist of only people".

B.8 SASD

34. What are the key words used by the SASD Methodology?

The following are the important key words which are used frequently by SASD Methodology.

Blackbox, control, Cohesion, Coupling, Data Structured Diagram alias Data Access Diagram, Data-Dictionary, Data Flow Diagram, Data Store, Data Structure, Factoring, Function, Functional Specifications, Implementation, Mini-specification, Modules, Process, Pseudo-Code, Sink, Source, Structured chart, Structured English, Structured Design Specification, Transaction, Transaction Analysis, Transform Analysis, etc..

35. Which of the above key words are not used accordance to their definitions given by IEEE?

Data Flow Diagram:

IEEE defines the term as: "A graphic representation of a system, showing data sources, data sinks, storage, and processes performed on data as nodes, and logical flow of data as link between the nodes" [34].

SD defines the term DFD as: "A graphic tool for depicting the partitioning of a system into a network of activities and their interfaces, together, with the origins destinations, and stores of data" [61].

Note:

1. In a DFD the external entities which may be involved in the System are identified first. This gives a sort of System boundary. This step is similar to that of JSD in which a list of entities is prepared.
2. It looks as though the IEEE and SD are using DFD for different purposes. The first one uses them for a graphic representation of a System and things related to its data, the second uses DFD for the partitioning of a System into networks of activities, to show what different modules will do and how they will interact with each other.
3. The term activity is used but not defined, perhaps it is used in a general sense.
4. Perhaps they mean the same thing but they are defining different aspects of a DFD. It needs the revision of the definitions, at least by the IEEE an independent body.

<u>Function:</u>

IEEE defines the term Function as: (1) "A specific purpose of an entity or its characteristic action". (2) A subprogram that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with sub-routine" [34].

The term Function is used but not defined explicitly. From their literature it is clear that they are using this word informally, in-consistently and in general terms which are used in finance or business e.g. ACCOUNT RECEIVABLE, ACCOUNTS PAYABLE, etc.. They also use the term Function for the tasks such as generating monthly pay roll, performing a mathematical function (e.g. calculation of square root, Basel Function, etc.). In SASD a Function is described by an imperative sentence, ideally having an active verb such as PRODUCE, COMPUTE, DELETE, etc.. The Functions are expressed like an order which can be easily understood. However, like other Methods/Methodology there is no explicit criteria to select a Function.

Functional Specification:

The term Functional Specification is defined by IEEE as: "A specification that defines the functions that a system or a system component must perform." [34].

SASD has not defined the term Functional Specification. However, it uses the term Functional Specification as the synonym for Functional Requirements which is defined as: "A precise description of the requirements of a computer system; includes a statement of the inputs to be supplied by the user, the outputs desired by the user, the algorithms involved in any computations desired by the user, and a description of such physical constraints as response time, volumes, and so on" [P-412, 20].

Note: The SASD term Functional Requirements as a synonym for Functional Specification differes from the IEEE defination of Functional Specification. The term only desiribes what the computer System is required to do, what are the required input/output, etc.. It is free from the physical consideration of how it is to do them. It is infact the re-statement of the problem in a manner which emphasises the data flow and ignores the how aspects. From the term Functional Requirements or Functional Specification it is not possible to know the characteristics of a function.

Implementation:

IEEE defines the term as: (1) "A realization of an abstraction in more concrete terms; in particular, in terms of hardware, software, or both". (2) "A machine executable form of a program, or a form of a program that can be translated automatically to machine executable form". (3) "A process of translating a design into code and debugging the code" [34].

SD defines the term Implementation as: "The activity of project during which the design of a System is tested, debugged and made operational" [61].

Note:

1. This term is used by SD to convert the output Design into code and test it. Under this heading activities of programming, coding and testing are covered. The scope

of the term differs from IEEE in the sense that SD does not involve the realization of an abstraction in terms of hardware.

2. SD supports incremental Implementation approaches over traditional approaches. The Methodology recommends adoption of any of the incremental approaches such as Top-down, Bottom-up, combined etc.. The criteria for selection of any of the options depends on the resources and the problem itself.

3. The Implementation of modules in SD can be a problem, because the Implementation activities i.e. coding, testing and debugging of modules involves iterations, refinements, re-arrangements of modules etc.. These activities become very expensive in terms of time and money involved, especially when the System has a large number of modules.

Module:

The term is defined by IEEE as: (1) "A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from, an assembler, compiler, linkage editor or executive routine". (2) "A logically separable part of a program" [34].

SD defines the above term as: (1) "A collection of program statements with four basic attributes; input and output, function, mechanics, and internal data" [61]. (2) "A contiguous (in actual contact) sequence of program statements, bounded by boundary elements, having an aggregate identifier" [20].

Note:

The two definitions seem to be very close to each other. In SD a Module is referred by its name, and can be called by other Modules. It differs from a program in the sense that an operating system can talk to a program but usually not to a Module in the program. Examples of a Module are; PROCEDURE in PL/1; SUBPROGRAM, SECTION, etc. in COBOL; FUNCTION in FORTRAN.

Process:

IEEE defines the term as: (1) "In a computer system, a unique, finite course of events defined by its purpose or by its effect, achieved under given conditions". (2) "To perform operations on data in process" [34].

SA defines the term as: "A set of operations transforming data, logically or physically, according to some process logic" [12].

SD defines the term as: "An activity on a data flow diagram that transforms input data flow(s) into output data flow(s)" [61].

Yourdon explains in his notes that a Process transforms data, i.e. it takes data and processes it. In a DFD it is represented by a bubble, inside the bubble is the name of the Process [18].

Note:

1. In general a Process may or may not perform a function i.e. some Process may be done without having any function hence achieving any result. The Process which performs a function, in SASD, is called a module. In other words a module in a program or a System is a Process which does some function. But all Processes are not necessarily modules.
2. SASD is using the concept of Process with limited scope. They are using this term only for transformation. The diagramatic representation of Process and transform is identical i.e. both are represented by upright rectangles having rounded corners or small circles. The symbolic representation of Process in SA is better than that of SD in which it is difficult to have legible writing within the circle. The upright rectangle having rounded corners in SA usually has three sections. Each section shows Process identification, description of function and physical location.
3. In SA the logic of a Process can be described by decision trees, decision tables and structured English.

4. Further in SASD other scopes of Processes are ignored. For example there may be some Processes which can be activited or performed without any data.
5. In SASD a Process may be a manual or automated activity, doing something, or a combination of manual and automated activities. It is a sequence of actions, operations, changes by using or creating data. In SASD Process (mini) specification does not show control, end of file indicator, error reporting.
6. The concept of Process used in by SASD is a special case of the 2nd. definition of IEEE.
7. The term function is used above in terms of their common use in SASD, in which it is described by an imperative sentence.

REFERENCES

References

1. Bell, T. and Thayer, T., "Software Requirements: are they really problem", Proceedings of 2nd International Conference On Software Engineering, San Francisco, pp. 61-68, 1976.

2. Fischer G. & Schneider M.,"Knowledge Based Communication Processes In Software Engineering", Proceedings of 7th. International Conference On Software Engineering, Orlando, Florida, USA., pp. 359, March 1984.

3. Freeman P.,"Why Johnny Can't Analyze", Systems Analysis and Design, A Foundation For The 1980's, pp. 323, Elsevier North Holland Inc., USA, ISBN:0-444-00642-7, 1981.

4. Longman New Universal Dictionary, Longman, ISBN: 0-582-55542-6, 1982.

5. De Marco T., Structured Analysis and System Specification, pp. 4, Prentice_Hall Inc., Englewood Cliffs, N.J. USA.

6. Brown G. L., "Logical Design Of Computer Based Information System", Systems Analysis and Design, A Foundation For The 1980's, pp. 182_83, Elsevier

North Holland Inc., USA, ISBN: 0-444-00642-7, 1981.

7. Wasserman A.I.,"The User of Software Engineering Methodology: An Overview", Information Systems Design Methodologies: A Comparative Review, Proceedings Of IFIP WG.8.1, 10-14 May 1982, The Netherlands pp. 591_628, North Holland Publishing Company, ISBN: 0-444-86407-5, 1983.

8. Mumford E. & Henshall D., A Participative Approach To Computer Systems Design, Associated Business Press London, ISBN: 0-85227-2219, 1979.

9. Mumford E. & Weir M., Computer Systems in Work Design_The ETHICS Method, Associated Business Press London, ISBN: 0-85227-2308, 1979.

10. Sarvari I.I., "Modles & Languages For Software Specification and Design", Workshop Notes: International Workshop on Models & Languages For Software Specification & Design, Orlando, Florida, USA, pp. 119, ISSN: 0225-0667, March 1984.

11. An Introduction To SADT, SofTech, Inc., Waltham, M.A., Document 9022_78, Feb., 1976.

12. Gane, c. & Sarson, t., Structured Systems Analysis, Englewood Cliffs, N.J. Prentice_Hall, ISBN: 0-13-854547-2, 1979.

13. Teichroew D. & Hershey E. A., "PSL/PSA: A Computer Aided Technique For Structured Documentation And Analysis Of Information Processing System", IEEE Transaction Of Software Engineering, pp. 41-48, Jan. 1977.

14. Verheijen G.M.A, & Bekkum J.V., "NIAM, An Information Analysis Method", Information System Design Methodologies: A Comprative Review, Proceedings Of IFIP WG. 8.1, 10-14 May 1982, The Netherlands, PP. 537_89, North Holland Publishing Company, ISBN: 0-444-86407-5, 1983.

15. Alford, M. W., "Software Requirement Engineering Methodology", Final Report_Volume I; Ballistic Missile Defence Advanced Technology Center, Huntsville, AL , August 1979.

16. Introduction To LSDM, Learmonth & Burchett Management Systems, 22 Newman Street , London W1P 3HB, Document Version 2, Issue 0, April 1985.

17. Teichroew D., Macasovie P., Hershey E. A. & Yamamoto Y., "application Of The Entity_Relationship Approach To Information Processing System Modelling", Entity_Relationship Approach To Systems Analysis And Design , P.P. chen (ed), pp. 15_38, North Holland Publishing Company, 1980.

18. Yourdon E., Structured Design Workshop, Edition 2.1, Yourdon Inc., 1133 Avenue Of Americas, NewYork, N.Y.10036, 1980.

19. Ned Chapin, "Graphic Tools In The Design of Information Systems", Systems Analysis And Design, A Foundation For The 1980's, pp 121_ 162., Elsevier North Holland Inc., ISBN: 0-444-00642-7, 1981.

20. Yourdon E., Constantine L.L., Structured Design, Yourdon Press, 1133 Avenue of Americas, New York, N.Y. 10036, ISBN: 0-917072-11-1, 1978.

21. Jackson M.A., Michael Jackson System Development, Prentice_Hall, International, ISBN: 0-13-880328-5, 1983.

22. Jackson K., Simpson H.R., "MASCOT", RRE., Technical Note No. 778, RRE. Procurement Executive, Ministry Of Defence, Malvern Worcs, Oct. 1975.

23. Brodie M.L. & Silva E., "Active And Passive Component Modelling: ACM/PCM", Information Systems Design Methodologies: A Comparative Review, Proceedings Of IFIP WG. 8.1, 10-14 May 1982, The Netherlands, pp. 41_91, North Holland Publishing Company, ISBN: 0-444-86407-5, 1983.

24. McAuto, Stradis: System Development Methodology

Product Description (undated) 20pp.

25. Lundeberg M., "The ISAC Approach To Specification of Information Systems And Its Application To The Organization of An IFIP Working Conference", Information Systems Design Methodologies: A Comparative Review, Proceedings of IFIP WG. 8.1, 10-14 May 1982, The Netherlands, pp. 173_234, North Holland Publishing Company, ISBN: 0-444-86407-5, 1983.

26. Liskov B. H., " A Design Methodology For Reliable Software Systems", Tutorial on Software Design Techniques, 3rd. Edition, Edited By Peter Freeman, A.I. Washerman, IEEE, pp. 65-73, 1980.

27. Aschim F. Mostue B.M., "IFIP WG. 8.1 Case Solved Using Sysdoc and Systemator", Information System Design Methodologies: A Comparative Review, Proceeding of IFIP WG. 8.1, 10-14 May 1982, The Netherlands, pp. 15_40, North Holland Publishing Company, ISBN: 0-444-86407-5, 1983.

28. Macdonald I.G. & Palmer I.R. , "System Development In a Shared Data Environment", The D2S2 Methodology, Information System Design Methodologies: A Comprative Review, Proceeding of IFIP WG. 8.1, 10-14 May 1982, The Netherlands, pp. 235_283, North Holland Publishing Company, ISBN: 0-444-86407-5,

1983.

29. Meyer B., "A System Description Method", Workshop Notes: International Workshop on Models & Languages For Software Specification & Design, Orlando, Florida, USA., pp. 42, ISSN 0225_0667, March 1984.

30. Jones C., "Software Development: A Rigorous Approach", Prentice Hall, 1980.

31. Hamilton M. and Zeldin S., "Higher Order Software_ A Methodology for Defining Software". IEEE Tranasactions on Software Engineering Vol. SE_2 No. 1 pp. 9_32, March 1976.

32. Series Z, Recommendations (Z .101 to 104) Specification and Description Language (SDL) Vol. VI-1. CCITT.

33. Cerchio L., "A System Design Methodology Based on SDL", SDL News Letter No. 4, pp. 19-20, November 1982.

34. IEEE. Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1982

35. Gerhart, S. Musser, D. Thompson, D. et al, "Overview of the AFFIRM Specification and Verification System", IFIP 80.

36. Enger N. L., "Classical and Structured Systems Life Cycle Phases and Documentation", Systems Analysis and Design, A Foundation For The 1980's, pp. 1_24, Elsevier North Holland Inc., USA, ISBN: 0-444-00642-7, 1981.

37. Biggs C. L., Birks E. G., Atkins W., "Managing The System Development Process", Printice-Hall Inc., Englewood Cliffs N.J. 07632, ISBN No.0-13-550830- 4.

38. Blum B., "Modles and Language For a Specific Application Class", Workshop Notes: International Workshop on Modles and Languages For Software Specification and Design, Orlando, Florida, USA, pp. 68-69, March 1984.

39. Ramamoorthy c. v. & Ma Y. N. "Design and Analysis of Computer Communication Systems", Systems Analysis and Design, A Foundation For The 1980's, pp. 452-460, Elsevier North Holland Inc., USA, ISBN: 0-444-00642-7, 1981.

40. O'Neill D., "Software Engineering Techniques Applied To The Systems Development Process", Systems Analysis and Design, A Founadtion For The 1980's, pp. 330-341, Elsevier North Holland Inc., USA, ISBN: 0-444-00642-7, 1981.

41. Freeman P., "The Context of Design", Tutorial on

Software Design Techniques, 3rd. Edition, pp. 2-4, IEEE Computer Society, 1980.

42. Tutorial, JSP & JSD: The Jackson Approach to Software Development , IEEE Computer Society, ISBN No. 0-8186-8516-6.

43. Mills H., "Top-down Programming In Large Systems", Debugging Techniques in large Systems. (courant Institute)

44. Hamilton M, and Zeldon, "Top-down, Bottom-up Structured Programming and Program Structuring", Charles Clark Darper Laborotary, M.I.T., December 1972.

45. Corner M.F.,"Structured Analysis and Design Techniques", Systems Analysis and Design, A Foundation for the 1980's, pp. 213-234, Elsevier North Holland Inc., ISBN: 0-444-00642-7, 1981.

46. M. U. Shaikh, "Techniques of Documentation of Computer Programs", M.Sc. Dissertation (partial), Department of Computational and Statistical Sciences, University of Liverpool, U. K., Nov. 1977.

47. Wolverton, R.W., "Software Life Cycle Management-Dynamics Practice", 2nd Software Life Cycle Management Workshop, Atlanta, Georgia, USA.,

pp. 21-29, 78CH1390-4C, August 1978.

48. Boehem, B. W., et al., "Prototyping Vs. Specifying: A Multi-Project Experiment", Proceedings of 7th. International Conference On Software Engineering, Orlando. Florida, USA, pp. 473-484, March 1984.

49. Muzzucchelli L., "Position Paper for the Workshop", Workshop Notes: International Workshop on Models & Languages for Software Specification & Design, Orlando,Florida, USA., pp. 196-197, ISSN: 0225_0667, March 1984.

50. Dijkstra, E. W., "Structured Programming", NATO Science Committee- Software Engineering Techniques, April 1970.

51. Dahl, O.J., Dijkstra, E.W., Hoare. C.A.R., "Structured Programming", Academic Press London, 1972.,

52. Wirth, N., "Program Development by Stepwise Refinment", Communication of ACM, 14(14): 221-227, 1971.

53. Maynard J., Modular Programming, New york: Petrocelli Books, 1972.

54. Brooks, F.P. Jr., "The Mythical Man-Month",

Reading, Mass.: Addison-Welsey Publishing Company, 1975.

55. McCain, R., "A Software Development Methodology for Reusable Components", Proceedings of the Eighteenth Annual Hawaii International Conference on System Sciences, Vol. II, Software, p. 319, USA, 1985.

56. Boeham, B.W., et al., "Characteristics of Software Quality", TRW Series on Software Technology vol 1, North-Holland, 1978.

57. McCabe, T.J., "A Complexity Metric", IEEE. Trans. on Software Engineering, Vol SE-2, No. 4, pp. 308-320, Dec. 1976.

58. Halstead, M.H., "Elements of Software Science", New york: American Elsevier, 1977.

59. Jones, T.C, "Measuring Programming Quality and Productivity", IBM. Systems Journal, Vol 17, No.1, 1978.

60. Kitchenham, B.A., "Measures of Program Complexity", ICL Technical Journal, 1981.

61. JONES, M.P., "The Practical Guide to Structured System Design", Yourdon Press, ISBN: 0-917072-17-0, 1980.

62. Chyou, S.C., "Structured Charts and Program Correctness Proofs", Proceedings of 7th. International Conference On Software Engineering, Orlando, Florida, USA, pp. 486-498, March 1984.

63. Olle, T.W., Sol, H.G., et al., "Information Systems Design Methodologies", A Comparative Review, Proceedings of the IFIP WG.8.1, 10-14 May 1982, The Netherlands, North Holland Publishing Company, ISBN: 0-444-86407-5, 1983.

64. Olle, T.W., Sol, H.G., et al., "Information Systems Design Methodologies", A Feature Analysis, Proceedings of IFIP WG 8.1, Working Conference on Feature Analysis of Information System Design Methodologies, York, Uk., Elsevier Science Publishers B.V., The Netherlands, ISBN: 0-444-86705-8, July 1983.

65. Maddison, R.N., et al."Information System Methodologies", Wiley Heyden, UK., ISBN 0-471-90332-9, 1983.

66. Tavendale, R.D., "A Technique For Prototyping Directly From a Specification", Proceedings of 8th. International Conference On Software Engineering, London, pp. 224-229, ISBN: 0-8186-0620-7, August, 1985.

67. Manley J.H., "Software Engineering Provisioning Process", Proceedings of 8th. International Conference On Software Engineering, London, pp. 273-284, ISBN: 0-8186-0620-7, August, 1985.

68. Belkhouche, B., "Compilation of Specification Languages as a Basis for Rapid and Efficient Prototyping", 3rd. International Workshop on Software Specification and Design, London, pp. 16, ISBN: 0-8186-0638-X August 1985.

69. Schach, S.R., "Prototyping, Design and The Early Cost Estimation Problem", 3rd. International Workshop on Software Specification and Design, London, pp. 212-213, ISBN:0-8186-0638-X, August 1985.

70. Weinberg, G.M., and Freedman, D.P., "Reviews, Walkthroughs and Inspections", IEEE. Trans. Soft. Eng. Vol SE-10, No.1, 68-72, Jan. 1984.

71. LBMS DP Training, Learmonth & Burchett Management Systems Ltd, 22 Newman Street London W1P 3HB, UK, 1983-84.

72. MASCOT and Management, Inter Establishment Committee on Computer Applications, Royal Signals and Redar Establishment, Malvern, Worcs, UK.

73. Jackson K, Moir C.I., "Parallel Processing in Software and Hardware-The MASCOT Approach", Sagmore Computer Conference on Parallel Processing, pp. 71-78, 1975.

74. Jackson K., Cloke J.A., "A MASCOT Software Standard",

75. Jackson K., "Evolution of MASCOT", System Designers Scientific, Pembroke House, Camberley, Surrey, Uk, June 1985.

76. Mumford E., "Job Satisfaction: a Method of Analysis", Personal Review, Vol 1, No. 3, Summer 1972.

77. Mumford E., "Participative Systems Design: Structure and Method, Systems, Objectives, Solutions 1, PP. 5-19, North Holland Publishing Company, 1981.

78. Denvir el at, "Report on the Study of an Ada Based System Development Methodology", Dept. of Industry, Uk, Sept. 1981.

79. Sol H.G., "A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations", Information Systems Design Methodologies, Proceedings of the IFIP WG. 8.1 Working Conference on Feature Analysis of

Information Systems Design Methodologies, York, UK., pp. 1-7, Elsevier Science Publishers B.V. ISBN: 0-444-86705-8, July 1983.

80. Brandat I., "A Comparative Study of Information System Design Methodologies", Information Systems Design Methodologies, Proceedings of IFIP WG. 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies, York, UK, pp. 9-35, Elsevier Science Publishers B.V., ISBN: 0-444-86705-8, July 1983.

81. Wasserman A.I., Freeman P., and Porcella M., "Characteristics of Software Development Methodologies", Information Systems Design Methodologies, Proceedings of IFIP WG. 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies, York, UK, pp. 37-58, Elsevier Science Publishers B.V. ISBN: 0-444-86705-8, July 1983.

82. Wasser A.I., Freeman P., "Ada Methodologies Concepts and Requirements", Software Engineering Notes Vol 8, Number 1, pp. 33-50, Jan. 1983.

83. Procella M., Freeman P., Wasserman A.I, "Ada Methodology Questionaire Summary", Software Engineering Notes Vol 8, Number 1, pp. 51-98, Jan. 1983.

84. Olive A., "Analysis of Conceptual and Logical Models in Information Systems Design Methodologies", Information System Design Methodologies, Proceedings of IFIP WG. 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies, York, UK, pp. 63-85, Elsevier Science Publishers B.V., ISBN: 0-444-86705-8, July 1983.

85. Iivari J., Kerola P., "A Sociocybernetic Frame Work for the Feature Analysis of Information Systems Design Methodologies", Information Systems Design Methodologies, Proceedings of IFIP WG. 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies, York, UK, pp. 87-139, ELsevier Science Publishers B.V., ISBN: 0-444-86705-8, July 1983.

86. Falkenberg E., Nijssen G.M., Adams A., Bradley L. "Feature Analysis of ACM/PCM, CIAM, ISAC and NIAM, Information System Design Methodologies, Proceedings of IFIP WG. 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies, York, Uk, pp. 169-189, Elsevier Science Publishers B.V., ISBN:0-444-86705-8, July 1983.

87. MASCOT Supplier Association, The Official Definations of MASCOT", RSRE, L-303 (s), St. Anderws Rd., Malvern, Worcs, Published in 1977.

88. Simpson H.R., "The MASCOT Approach To Software Design and Implementation", Ph.D. thesis, Dept. of Computong and Control, Imperial College of Science & Technology, London University., Nov. 1975.

89. Gilles Rigal, "Generating Acceptance Tests From SADT/SPECIF", IGL, 41, Rue De La Chaussee D'antin, 75009 Paris, France, August 1985.

90. Gilles Rigal, "Simulation With SADT", IGL, 41, Rue De La Chaussee D'antin, 75009 Paris, France, August 1985.