Postprint of article in *Australian Computer Journal* **14** (3): 99–109 (1982)

# A Review of System Development Systems*

T.H. Tse and L. Pong
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
Email: thtse@cs.hku.hk

**ABSTRACT**

The requirements for a system development system are defined and used as guidelines to review six such systems: SAMM, SREM, SADT, ADS/SODA, PSL/PSA and Systematics. It is found that current system development systems emphasise only validation and user verification. They can perform relatively little on automatic file optimisation, process optimisation and maintenance.

Keywords and phrases: requirements specifications, software development, software engineering, systems design, systems development.

CR Categories: 4.19, 4.22, 4.6 [D.2, D.2.l, K.6.1, K.6.3].

## 1. INTRODUCTION

In the early days of stored program computers, the cost of software made up a mere 15 per cent of the total cost of information systems. But software cost has been escalating ever since and is currently estimated at about 90 per cent of the total, as shown in Figure 1 taken from Boehm (1976). It is more alarming to note that more than two-thirds of the money is spent on the maintenance of existing software and only one-third on new developments.

The high cost of software in information systems can be attributed to the following:

(a) Since the days of ENIAC and EDVAC, computer hardware has evolved from the first generation of vacuum tubes to the fourth generation of very large scale integration. Software development methodologies also went through a similar evolution. The first generation was simply a consolidation of conventional techniques previously used for manual systems. New techniques were developed in the second generation specifically for computer systems. In the third generation, some parts of the development were automated. Finally, in the fourth generation, we see fully automated system development systems. Unfortunately, as Couger (1973) pointed out, the software evolution has been lagging behind the hardware evolution by one full generation. Most systems are still being developed using second and third generation techniques.

(b) Because of the belated use of computer aids, information system development in practice has been a manual process. Hence the adverse effects of manual systems also apply, such as the

escalation of manpower cost and control problems of large projects.

(c) In the absence of comprehensive computer aids, errors made during the analysis stage cannot easily be identified. The effect is multiplied when the system is implemented together with the error. One study by IBM (Fagan, 1974) has revealed that the cost of correcting an error after implementation is almost a hundred times that of correcting the same error during the analysis stage.

In view of the above cost escalation, research workers have been pursuing the concept of automated "system development systems", or *sds* for short.

A few surveys have been published in the line of system development systems, but the emphasis is often on one aspect of it, namely requirements analysis. The most notable examples are Teichroew (1970), Burns (1974), Ramamoorthy and So (1977) and Jones (1979). Little has been reviewed on the full aspects of system development systems.

This paper attempts to

(a) Set out the comprehensive goals for a system development system, and

(b) Give a review and evaluation of six most fully developed system development systems.

## 2. GOALS

In this section we will identify the goals for a system development system, so that a framework of criteria can be established before we evaluate the current practices in sds.

### 2.1 Validation

For a large complex system, the requirements specification may be very large. It is impractical to leave all correctness checking as a manual process. Further, every time the requirements are changed, the correctness of the specification will need to be confirmed again. One function of the sds is therefore to validate the specification. We can separate the validation function into the following areas:

#### *2.1.1 Completeness*

Waters (1979) has given a list of 77 "facts" that should be incorporated into a specification. The system development system should be able to accept these facts as parameters, and verify that the vital parameters are present at the appropriate places.

One common feature used to conceal the incompleteness of the requirements specification is the generous use of memos or comments, whereby the user can put in anything in natural language. But such memos cannot be analysed by the sds or compiled into program code.

#### *2.1.2 Continuity*

In the flow of information, we must make sure that:

- data items must have been input from a source or derived from other data items,
- data items input or derived must be traceable to some use or destination,
- data items must not be defined in cycles,
- data items that are used in a subsystem must not call for data items outside the subsystem implicitly.

Precedence analysis is required to check the continuity of data flow. It was first introduced by Langefors (1963) and further developed by others such as Waters (1976 and 1977).

*2.1.3 Consistency*

The need to check the logical consistency of a requirements specification has long been known since the second generation of information systems methodology, and has been incorporated into numerous student texts. (See, for example, Fergus, 1969.) In essence, the sds should pin-point the following:

- Inconsistencies among different parts of the specification;
- Parts of the requirement that have been referred to but not specified;
- Where control checks have been built in, any discrepancy between the control figures and the actual specification.

*I.1.4 Redundancy*

The sds must be so designed that

- there will be no need to duplicate the requirement in different parts of the specification, such as common areas in different modules;
- if duplicated information has been presented by mistake, it should be brought to the attention of the user for scrutiny and possible correction.

## 2.2 User Verification

One criterion for a good requirements specification is that the user must be able to review the specification and verify whether it really represents their needs. This can be achieved in several ways:

*2.2.1 Use of Natural Language*

This would enhance the understanding of the specification. There are, unfortunately, few systems which can accept natural language as input. Further, natural languages may be subject to different interpretations.

*2.2.2 Generation of Documentation*

To avoid problems in compilation, pseudo-code is used as input to the sds. Since pseudo-code is compact and difficult to read, it would be useful for the sds to generate narrative documentation to ease the understanding of the specification. Ambiguities, however, may still appear in the narratives thus generated, so that the user will have to trace back to the original pseudo-code.

*2.2.3 Graphic Aid*

It has been generally agreed that graphic presentation of complex material is much more comprehensible than the narrative counterpart. The reasons can be summarised as follows:

Graphics are in two dimensions while narratives are in one dimension. The former gives an additional degree of freedom in presentation.

- The person reading graphics can do so selectively, depending on the level of details he wants. If they read a narrative description, they have to do so linearly.
- There is a limit in the number of concepts one can reasonably hold in short term memory in the human brain. (This number is believed to be $7 \pm 2$ by Miller, 1956.) The person reading graphics can start off generally and go into details after some degree of familiarisation. If they read narrative, they have to start off with details and abstract the skeleton concepts afterwards.

In addition, graphic input usually provides a bounded rationality which is useful for such validation aspects as completeness, continuity and consistency.

An ideal requirements specification would therefore be documented in graphics, which is then machine processed. Failing that, if a requirements specification is in pseudo-code, it must be converted into graphic output for verification and correction.

#### *2.2.4 Prototype System*

The user may not be able to completely specify their requirements at an early stage without having a "feel" for the outcome of the system. The sds should therefore allow the user to partially specify their needs, compile it into a prototype system, and feed it back to the user. Given the prototype output, the user can then refine their requirements through modifications and/or provision of additional details. Jones (1979), for example, compares prototype systems favourably with other defect detection methods, as shown in Table 1.

### 2.3 File Design and Optimisation

During the physical design phase, the systems analyst has to design files to suit the given hardware environment and processing requirements. They are faced with an over-abundance of choices, which grow exponentially with the complexity of the system and hence become unmanageable to the human mind. As a result, they often base their choices on simple "rules of thumb", also known as "experience". The solution given by these simple rules are workable but usually far from optimal, so that systems designed are more expensive to operate than necessary. Some authors (such as Waters, 1972) even come to the conclusion that such simple rules of thumb are so ineffective that we might as well disregard them.

The system development system, in order to produce better file design, should include file optimisation modules. There are three approaches available.

#### *2.3.1 Simulation Models*

Simulation techniques were mainly used in the earlier models for the evaluation of file organisations. Examples are Senko et al. (1968) and Cardenas (1973). Since each model assumed a specific file organisation, it was impossible to obtain an overall optimum unless multiple models were run and compared.

#### *2.3.2 Analytic Models*

The problem of physical file design can be expressed in mathematical programming terms as follows:

Given constraints such as hardware configuration and probabilistic details of data, we want to determine the factors such as file structure, access method, overflow mechanism, etc., so that the cost of data retrieval and file maintenance is minimised, where the cost is a function of response time and storage space.

We note, however, that the problem has the following characteristics:

(a) For a given set of factors, the cost can be determined analytically, but it is a non-linear function of the factors.

(b) For realistic situations, the number of [actors is large. Severance and Duhne (1976), for example, list the following factors for the selection of hashing algorithm alone:

- identifier transformation
- overflow technique
- overflow area
- initial loading order
- bucket size

- loading factor

(c) Further, each one of the factors allows a large number of discrete choices. Martin (1977), for example, lists the following choices for identifier transformation:

- mid-square method
- dividing
- shifting
- folding
- digit analysis
- radix conversion
- Lin's method
- polynomial division

For each discrete choice, we have to define a separate variable with {0, 1} values.

The minimisation problem therefore becomes one in non-linear integer programming with a large number of zero-one variables.

Yao and Merten (1975) simplify the problem by concentrating only on the average characteristics of file organisations, based on a single analytic model designed by Yao (1977). The number of variables then becomes manageable. Using this as a first approximation, the number of subsequent choices will be limited. The detailed structures of the file can then be worked out by simulation.

One drawback of this method is that the first approximation can only give a crude result, so that we may be making a fine adjustment based on a wrong decision.

Other techniques in file optimisation include the approximation of an integer programming model by a continuous model, and the method of branch and bound to reduce the number of searches to a manageable size.

#### *2.3.3 Heuristics*

We have already seen that over-simplified rules of thumb are not satisfactory. But some researchers (such as Severance and Duhne, 1976, and Severance and Carlis, 1977) have derived heuristic rules from mathematical models. Such rules are more realistic and can be implemented into the sds for a near-optimal file design.

### 2.4 Process Design and Optimisation

One of the primary functions of an sds is to "compile" the requirements specifications into program code. It is all too easy to generate program code by brute force. So an sds must be able to optimise the processing. (This is known to some authors as logical design as distinct from the physical design covered in the previous section of the paper.)

Processing requirements can be divided into two categories: *transportation of data items* (that is, input and output) and *derivation of data items* (that is, computations). In information systems, however, the number of derivations is small compared with transportations. Hence for purposes of optimisation, we can concentrate on the transportation aspect.

Transportation volume can be reduced as follows (Alter, 1979, and Severance and Lohman, 1976):

(a) Vertical aggregation of processes:

Two sequential processes having a file as an intermediate buffer may be combined to reduce input/output volume.

(b) Horizontal aggregation of processes:

Processes reading the same file may be combined to reduce the input volume.

(c) Aggregation of files:

Files generated by the same process may be combined to reduce the output volume.

(d) Use of differential files:

Data items having different frequencies of access are candidates for separation into two files.

A more general treatment can be made using general net theory (Genrich et al., 1980). The information system is expressed as a bipartite graph known as a net, having two types of nodes called states (corresponding to our files) and transitions (our processes). A topology is defined, and hence the concepts of open sets (our process being an extreme example) and closed sets (our files). Morphisms can then be defined between nets. The aggregations and separations of files and processes are specific cases of net morphisms.

Given the very large number of possibilities of net morphisms, the sds is faced with the task of selecting the optimal one. It is even more difficult in this case to apply the mathematical programming models discussed in the previous section. Alter (1979) proposes the use of an iteration method. The optimisation problem is divided into two phases: *file optimisation* and *process optimisation*.

An initial solution is input to the first phase to enable a search for an optimal file design. This design is input to the second phase in search for an optimal process design. The latter then goes back to the first phase for an improvement of the files. The iteration process is repeated until any more improvement becomes insignificant. Within each phase, we can either make use of mathematical programming, or build a smaller iterative loop. Alter, for example, suggests the latter, making use of the steepest ascent method.

### 2.5 Maintenance

Modifications to an information system are frequently necessary due to changes in the environment, technology or user needs. If the information system was created by an sds, there are three approaches to handle its maintenance:

(a) Change the requirements manually and re-input the entire specification to the sds. A new information system is thus generated.

(b) The sds accepts amendments to specifications of requirements. A requirements analysis phase produces feedback on the significance of the changes. After user verification, the sds produces a new requirements specification for input to the design and optimisation phase. A new information system is thus generated.

(c) The sds accepts amendments to specifications of requirements and produces feedback on its significance, as per (b) above. But instead of producing a new system, only the affected parts of the information system are changed.

The following considerations must be made when we decide on the approach to be adapted by an sds:

- There should be a record of the modifications made to an information system, for the purposes of audit and control. Method (a) does not provide this facility.
- Method (b) appears simple and neat for the user, but as Teichroew (1971) has pointed out, ‘‘the cost ... would be prohibitive if every change in a Problem Statement required a complete re-run of the whole system’’.

- Further, a re-run of the optimisation phase, as proposed in (b), may result in drastic changes in the file organisation methods of the system. Thus the old and new file structures may become incompatible.
- In depth studies are still required before method (c) can be implemented. Also this method may actually produce a sub-optimal solution since only part of the sds is re-run.

In practice method (b) is more suitable for major changes and method (c) for minor amendments.

Another aspect to consider for maintenance is the portability of the information system. The sds preferably should be a pre-compiler that accepts as input a requirements specification and generates as output one or more programs in a high level language. This alleviates the need to run the sds again when there is a change in the hardware configuration.

A further advantage is that programmers can change the output program directly if there is a bug in the sds, or if some additional requirement is out of the scope of the sds. It should be noted, however, that once programmers are allowed to tamper with program code, they tend to "short circuit" every maintenance job. They go directly to the program code for modifications without bothering to re-run the sds. This leads back to the usual control problem of manually produced information systems: that the specification does not agree with the programs.

One solution is for the sds to generate a hash total of all amounts and figures in the requirements specification, and to reconcile it against a similar hash total in the programs. If the hash total of the programs is changed while that of the specification remains unaffected, it means illegal alterations to the programs have taken place.

## 3. CURRENT STATE

In this section we will study six system development systems with a view to see how much they have achieved in the goals we have set out. SAMM, SREM and SADT are chosen because they are the latest developments. ADS, PSL/PSA and Systematics are chosen because they are the pioneers in sds, they are still popular, and there have been new developments lately, such as ADS/SODA and META/GA.

### 3.1 Systematic Activity Modelling Method (SAMM)

SAMM — a modelling method based on the Human Directed Activity Cell Model — has been developed by the Boeing Computer Services Company (Peters, 1978, Lamb et al., 1978 and Stephens and Tripp, 1978). It utilises a word-graphic language, which is a combination of the more prominent features of narratives, graphics and graph theoretic notations. The representation scheme of SAMM consists of a labelled tree, activity diagrams and condition charts. The labelled tree structure, as shown in Figure 2, provides hierarchical decompositions of the system and an index structure describing the context of activity diagrams in the system.

Activity diagrams, with the fundamental building blocks of activity cell and data flow, portray the relationships of activities and dataflows of the system. An activity diagram, as shown in Figure 3, is a flow diagram with a network of rectangular boxes representing activities, and arrows representing dataflows. Moreover, it contains a data table giving narrative description and decomposition trace of the data involved. The decomposition trace provides a hierarchy of data. The node name identifies the context of the diagram in the system. The number of activity cells in each diagram is restricted to six to conform to the principle that "the span of absolute judgement and the span of immediate memory of humans is in the vicinity of seven items" as stated by Miller (1956).

The activity-data network in each activity diagram corresponds to a directed graph. Therefore, analysis such as connectivity and reachability can be performed to give insight into the consistency of the specification.

Associated with each activity diagram is a condition chart. It describes the input and system/activity state requirements for the production of output.

A total system description in SAMM is thus an interrelated set of diagrams with a hierarchical structure. Each layer of diagrams in the tree structure represents a semantic interpretation of the system at a certain level of abstraction.

An automated tool, SAMM Interactive Graphic System (SIGS), is developed to implement SAMM. The functions of SIGS include model generation, model editing, model display, verification, report generation and model status control (see Figure 4).

Various analyses can be performed on the SAMM model input to the SIGS. These include syntax analysis to ensure that the input conforms to SAMM methodology: decomposition analysis to ensure consistency between a parent diagram and a child diagram, and between two child diagrams; data flow analysis; and global analysis which determines redundancy in the model. The tree structure is checked for connectivity and reachability. Diagnostic reports and documentation of selected subsets of the model can be generated by using the report generation facility.

SAMM possesses many desirable features of an sds — graphic representation, multilevel refinement, machine processability, centralisation of information and consideration for bounded rationality. It has, however, remained a requirements analysis system and has not tackled the aspects of file and process design, optimisation and maintenance.

A SAMM activity diagram can be regarded as a special type of data flow diagram (DeMarco, 1979), one without the expression of files. It can therefore be implemented *manually* using structured design methodologies (Stevens et al., 1974, and Yourdon and Constantine, 1979). But the SAMM language does not include such features as performance requirements, so that automated design and optimisation would not be possible without an extension of the language itself.

### 3.2 Software Requirements Engineering Methodology (SREM)

SREM — a computer-aided methodology for the development of ''no-man-in-the-loop'' real-time software — has been developed by the TRW Defense and Space Systems Group (Alford, 1977 and Bell et al., 1977). The methodology consists of the Requirements Statement Language (RSL) (Bell and Bixler, 1976) and the Requirements Engineering and Validation System (REVS). An overview of SREM is shown in Figure 5.

The fundamental approach of SREM, based on the fact that paths of processing are invariant over any process design (Alford and Burns, 1976), is to specify software requirements in terms of flows through the system. The paths of processing, each representing a sequence of operations connecting the arrival of the input message to the termination of its processing, are organised into Requirements Nets (R-Nets) for understanding and analysis. Each Requirements Net represents the network of the processing steps in response to a given type of stimulus. An example is shown in Figure 6.

Provisions for stating performance requirements are made by the notion of validation points in the R-Nets. At such points, performance characteristics are defined and verified against actual data.

The description of a stimulus-response sequence of a system in the form of a Requirements Net can be further detailed in a top-down manner. An ALPHA (processing step) in a Requirements Net can be expanded into another flow graph with lower level ALPHAs and the original ALPHA is replaced by a SUBNET in the parent flow graph. This decomposition process can be continued until any further detailing will force unnecessary constraints on system design. Requirements Nets can be input to the REVS explicitly through an interactive graphic tool, or implicitly through the structure aspect of RSL statements.

RSL has four types of primitives: structures, elements, binary relations between elements and attributes of elements. The structures expose the flow portion of the requirements. They are the

images of the Requirements Nets projected on to a one-dimensional space. Elements, relations and attributes deal with the non-procedural portion of the requirements. These primitives can formulate every concept in RSL. The structures are fixed to provide a rigid framework for communication. However, the non-procedural aspects of the RSL are extensible to suit particular applications and future needs.

REVS is comprised of the RSL translator, the Abstract System Semantic Model (ASSM) and a set of analysis tools. The RSL translator is obtained by employing a compiler-writing system so that changes in RSL can be easily and effectively accomplished. The ASSM is a relational database for maintaining information on software requirements and the concepts used to express the requirements. Extensions in concepts can be processed in the same way as RSL statements by the RSL translator and are ready for use as soon as they are entered into the ASSM. The ASSM also provides a decoupling between the RSL and the analysis tools so that modifications on either end can be made independently. Analysis tools have been developed to perform analyses on the information stored in the ASSM. They include static analysis tools —– which check the correctness and consistency of the R-Net structure and data flow; dynamic analysis tools —– which generate discrete functional and analytic simulators semi-automatically to check dynamic system interactions; and a flexible generalised extractor package for documentation and special reports.

SREM represents a very sophisticated requirements definition methodology. It is designed to fit into a complete system development framework (Davis and Vick, 1977). A process design engineering methodology has been reported, but interface smoothness and the stage of development are not known. Details of file design, optimisation and maintenance in the complete development framework are also not available. RSL has a graphic representation and allows multilevel refinement, but it has no bounded rationality consideration. Consistency and continuity checks can be done automatically. Performance requirements are formally stated by the use of validation points. Semi-automatic simulation is provided to analyse the dynamic behaviour of the system being developed. This provides the analyst and the user with a clear perception of the system at an early stage.

### 3.3 Structured Analysis and Design Technique (SADT)

SADT —– a general modelling method that can be applied to a wide range of systems —– has been developed by SofTech (Ross, 1977, Ross and Schoman, 1977 and Dickover et al., 1978) based on the concepts of structured analysis of Ross (1980). It is basically a structured thought and decomposition discipline with a graphic means of expression. The structured and disciplined way of thinking and decomposition is established and applied before thoughts are expressed by the graphic tool.

The system model behind SADT consists of ‘‘things’’, ‘‘happenings’’ and their relationships. Therefore, each SADT model consists of two dual decompositions —– data decomposition and activity decomposition. Each of these decompositions uses the same graphic tool.

The fundamental building block of the SADT graphic notation is the four-sided box shown in Figure 7. Each of these boxes conveys certain details of the system being described. The INPUT, OUTPUT and CONTROL arrows specify interfaces to other boxes. The MECHANISM arrow shows the support to accomplish the transformation represented by the box.

An SADT system description consists of an interconnected set of diagrams in which interrelated boxes and arrows provide a disciplined framework for the embodiment of any natural or artificial language expression chosen for a particular application. This framework obeys rigorous semantic and syntactic rules so that the interpretations of the embedded language expressions are restricted. The top level diagram shows the overall network structure of the system, with boxes and arrows showing the components and interactions. Each box may be further decomposed into a separate diagram with another boxes-and-arrows network, as shown in Figure 8, provided the detailed diagram represents exactly the same part of the system as the original box. Decomposition of boxes can be carried on at all levels, and as a result a top-down hierarchical structure is established.

Arrows in SADT diagrams do not stand for control flow, but represent constraints. Precedence relationships, however, do exist because a box at the beginning of an arrow must precede that at the end. These precedence relations may imply parallelism. SADT diagrams represent all these implicit parallelisms unless the system designer explicitly decides to impose sequencing constraints by using the SADT activation rules.

A complete SADT description of a system may consist of d set of interrelated SADT models. Each SADT model consists of a hierarchical set of diagrams that describe a subject from an identified viewpoint, for a particular purpose and within a specific context. The viewpoint determines what is to be described, the purpose determines how the subject is to be described and the context enforces proper understanding of what is described. In other words, these attributes bound and limit the amount of the subject that can be exposed and the way it is structured. The MECHANISM arrows provide the means to connect models having different orientations of viewpoint, purpose and context.

As a SADT model is a tree-like hierarchy, a node index is provided for each model so that the corresponding context for a particular diagram can be easily determined.

SADT provides a graphic means of expression and multilevel refinement of problem to aid understanding. Formal means to express performance information are not provided, nor are strict rules to analyse the specification for desired properties such as consistency and completeness. Moreover, as the complexity of the system increases, it is difficult to handle the technique manually. To improve the situation, Ross (1977) has suggested that SADT can ‘‘become machine-readable in a very straightforward manner’’ and hence extendible to include the validation facilities of the PSL/PSA system. No result has yet been published.

It should be noted, however, that SADT is not designed to be mapped on to an automatic development system. The fundamental concept of ‘‘omitting the obvious’’ in SADT, for instance, is only suitable for manual development. There is, therefore, no guarantee of a smooth interface with automatic design and optimisation.

### 3.4 ADS/SODA

ADS (Lynch, 1969 and NCR, 1969) was an internal standard of NCR and subsequently released for public use. It was originally intended to be a manual procedure. However, automation of its use has been reported (Couger, 1973 and Nunamaker et al., 1976).

An ADS system description is made up of five interrelated forms called RICHL (ritual): ***R***eport definition form, ***I***nput definition form, ***C***omputation definition form, ***H***istory definition form and ***L***ogic definition form.

Based on the notion that system development should be results-oriented, ADS system description starts with the definition of all systems output. It is then completed by descriptions of system input, computations, historical data retained in the system for a period of time, and the accompanying logic that will be used to derive the output.

Information in these forms is interrelated by the flow of data. Linking is made possible by assigning unique names to data elements and the backward referencing of each data element to its information source. These references are achieved by the use of the 3-tuple (Definition type, Page number, Line number) for each line on every ADS form. The data elements are therefore chained from output to input

ADS has been incorporated into the System Optimisation and Design Algorithm (SODA) (Nunamaker, 1971) to form an integrated computer-aided methodology for the development of a financial management system (Nunamaker et al., 1976). The methodology consists of ADS, SODA Statement Language (SSL), ADS analyser, SODA Statement Analyser (SSA), SODA Generator of Alternatives (SGA) and SODA Performance Evaluator (SPE).

SSL statements are used to provide design parameters and performance requirements not available in the ADS description.

The ADS description and SSL statements are analysed and validated by the two analysers. This analysis phase produces a series of summary reports including: a data dictionary, indices to all data elements and processes, incidence matrices of data elements required by each process, precedence matrices of data elements and processes, and graphical displays of the input ADS forms.

The output of the analyser and a statement of the available computing resources, hardware and utility programs are accepted by the SGA to analyse alternative hardware and software resources with respect to a specific design generated by the SGA. The output is a set of specifications of alternative designs stating the necessary CPU, core size, program structure and data structure.

SPE optimises feasible designs to improve system performance. It is made up of a series of mathematical programming models and timing routines. Its functions include optimisation of the blocking factor for files, determining the number and type of auxiliary memory devices, allocation of files to memory devices and generation of an operation schedule.

The ADS/SODA integrated system deals with both the analysis and design phases of the system development cycle. Requirements specification can be mechanically processed to ensure consistency and continuity. However, the ADS forms are often incomprehensible to the user and there is no hierarchical decomposition strategy to tackle complex problems. No graphic documentation is provided to facilitate understanding. Transition into the design phase is straight-forward. Optimisation of files and program structures are performed by the SPE, but the optimisation of program structures may cut across functional boundaries and may lead to maintenance difficulties. This optimisation must therefore be constrained. Moreover, SODA is restricted to the design of batch processing systems, sequential auxiliary storage organisation, the specification of linear data structures, and the selection of a single CPU. The designs generated are machine dependent as a particular design is based on a particular choice of hardware.

### 3.5 PSL/PSA and META/GA Systems

PSL/PSA –– a computer aided system for systems requirements documentation and analysis –– has been developed by the ISDOS project of the University of Michigan (Teichroew, 1976 and Teichroew and Hershey, 1977). The system consists of the Problem Statement Language (PSL), the Problem Statement Analyser (PSA) and a database for maintaining information of the system being developed, as shown in Figure 9.

The Problem Statement Language is a relational, non-procedural and machine processable language. It has well-defined syntax and semantics, and is designed for systems description. The underlying system model is the entity-relationship-attribute model. It can be described as a set of objects, their properties and binary relations between these objects. Consequently, PSL statements are object-relationship-object associations. Systems descriptions in PSL are classified into: system input/output flow, system structure, data structure, data derivation, system size and volume, system dynamics, system properties and project management. They are processed by the Problem Statement Analyser and stored into the PSA database.

The Problem Statement Analyser is a collection of computer software developed for processing and analysis of the PSL statements, and the management of the database information. Lexical, syntactic and semantic analyses are performed before the PSL statements are entered into the database. Complementary relationship statements are generated by PSA and entered into the database. Once entered, a statement can be expanded or deleted without major change to other statements. PSA can perform: data definition analysis; static analysis, which checks the consistency of the input statements; dynamic analysis, which determines dynamic relationships among input, output and timing consistency of processes; and volume analysis. Documentation and reports can be produced by PSA interactively or in batch mode. The reports produced can be classified as database

modification reports which deal with changes and diagnostics; reference reports which present PSL information in various formats; summary reports which summarise information according to several relationships; and analysis reports which present the results of the aforementioned analyses.

PSL/PSA was not originally designed to fit any particular system development framework. Its success therefore depends on how well it suits a chosen methodology. As the number of system development methodologies is continuously increasing, it is unlikely that the PSL/PSA will fit well into every one of them. Attempts, for example, have been made to incorporate PSL/PSA into SODA, but it was found that enhancements of PSL were necessary to include features of ADS and SSL (Nunamaker et al., 1976). The META/GA system is designed to remedy the problem.

The META/CA system (Teichroew et al., 1980 and Yamamoto, 1981) consists of the META system and the Generalized Analyser, as shown in Figure 10.

The META system, based on an entity-relationship-attribute model, takes a formal description of the PSL and automatically generates the language processor using a table-driven generalised software and a language reference manual. System descriptions can now be formulated in the particular PSL and manipulated by the Generalized Analyser in a similar way to that of the PSA. META/GA has been successfully applied to a number of methodologies such as Composite Design, Rational Design Methodology and Jackson Methodology.

PSL/PSA is one of the most widely used and accepted requirements definition systems. Recognising the shortcoming of the one-dimensional nature of PSL, a set of graphic reports can be generated automatically for user verification. However, as the source PSL statements are not verified by the user, one more pass may be needed. Moreover, there is no facility to trace back from the graphic reports to the source PSL statements. The non-procedural PSL allows multilevel refinement and is machine processable for static correctness. Centralisation of information is achieved via the database. Drawbacks of the system include: no formal means to state performance information, no aids to provide early visibility into the target system and no guarantee to fit well into a particular system development framework. The last defect is remedied by the development of the META/GA system.

### 3.6 Systematics

Systematics –– a language designed for analysing problems and specifying requirements –– was established by Grindley (1966, 1975 and 1979). It is built on an information algebra with the following basic concepts:

(a) Item –– Defined as ‘‘the smallest collection of signals which plays a separately definable part within the control system’’, an item is the most fundamental building block of the system.

(b) State –– A state is a particular occurrence of an item.

(c) Data Set –– A collection of all items playing the same role in the system.

(d) Primary Identifier –– Data set *A* is a primary identifier for data set *B* if a given state in *A* identifies one and only one state in *B*.

(e) Secondary Identifier –– Data set *A* is a secondary identifier for data set B if a given state in A identifies a set of states in B.

(f) Given Item –– A given item has its states submitted directly to the system.

(g) Derived Item –– A derived item has its states computed by the system.

(h) Information Set –– A collection of related items having the same primary identifier.

(i) Input and Output Sets –– An input set is an information set which is supplied to the system from outside. An output set is an information set that is used to notify states of items to outside. They

are the input and output records in physical terms.

(j) Trigger –– An input set that causes an output set to be produced.

(k) Effective Time –– A data set may vary its state over time. In this case there is an identifier of time implied. It is known as effective time (ET) in Systematics.

(l) Discrete and Continuous Identifications –– In discrete identification, a single state of a data set can identify other data sets. In the continuous case, a range of states is required for identification.

(m) Time Substitute –– An identifier which increases serially with time can be used as a time substitute in cases where it is impractical to specify time directly.

These fundamental concepts are used to construct specifications. Construction starts off in output sets and works its way to input sets.

An output set is defined by a Systematics sentence, which consists of three parts: the trigger, the output items, and the identifiers for each output item. Any output report or user enquiry is thus specified by a Systematics sentence. For users not familiar with Systematics, however, an output definition form is provided, so that the syntax of Systematics sentence becomes transparent to them.

To reduce the troublesome work of specifying the primary identifiers for all items on the output sets, a primary identification dictionary is employed, leaving only the non-primary ones in the output definition. The primary identification dictionary is in the form of a matrix.

A derivation dictionary is constructed to give the formulae for all items that are computed within the system. These formulae also provide an identification chain linking each component of the formulae with other formulae or other dictionaries.

Items must either be input or derived. Any item not included in the derivation dictionary must therefore be entered into the input dictionary. The latter is in a simple grid form showing the given data sets against the input sets. This dictionary facilitates the backward tracing of output to their given origins.

A graphic convention for presenting Systematics specifications has been developed correspondingly. An example is given in Figure 11.

Systematics is meant to be a manual sds to ease requirements analysis, file design and process design. It provides a well-defined methodology to determine the output requirements and hence the input and derivations. Though the graphic convention can be used to aid understanding, there is no hierarchical decomposition strategy.

Despite the humble remark that "it is not intended to inhibit the development of Systematics by providing it with a compiler" (Grindley, 1966), one of the authors (Tse) has been informed by Grindley that a Systematics compiler has already been written. Since published information is not yet available, details of optimisation and maintenance are not known.

## 4. CONCLUSION

In this paper we have drawn up the goals or the "requirements" of a system development system. They fall into five categories: validation, user verification, file design and optimisation, process design and optimisation and maintenance.

Six of the most popular sds have been chosen for review and evaluation. They are SAMM, SREM, SADT, ADS/SODA, PSL/PSA and Systematics. A summary of findings is given in Table 2.

It has been found that most of the sds emphasise only the validation aspect of the full system development spectrum. In addition, some of the systems provide user-friendly verification aids such as graphic input or graphic feedback. Relatively little work has been done in providing automatic aids in the areas of file optimisation, process optimisation and maintenance.

## 5. ACKNOWLEDGEMENTS

The authors are grateful to Dr C.K. Yuen and Mr S.W. Ho for their invaluable suggestions.

## 6. REFERENCES

M.W. Alford (1977): ''A requirements engineering methodology for real-time processing requirements'', *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, pp. 60−69.

M.W. Alford and I.F. Burns (1976): ''R-nets: A graph model for real-time software requirements'', in *Proceedings of the Symposium on Computer Software Engineering*, J. Fox (ed.), Microwave Research Institute Symposium Series, Polytechnic Press, Polytechnic Institute of New York, Brooklyn, NY, Vol. 24, pp. 97−108.

S. Alter (1979): ''A model for automating file and program design in business application systems'', *Communications of the ACM*, Vol. 22, No. 6, pp. 345−353.

T.E. Bell and D.C. Bixler (1976): ''A flow-oriented requirements statement language'', in *Proceedings of the Symposium on Computer Software Engineering*, J. Fox (ed.), Microwave Research Institute Symposium Series, Polytechnic Press, Polytechnic Institute of New York, Brooklyn, NY, Vol. 24, pp. 108−122.

T.E. Bell, D.C. Bixler, and M.E. Dyer (1977): ''An extendible approach to computer-aided software requirements engineering'', *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, pp. 49−60.

B.W. Boehm (1976): ''Software engineering'', *IEEE Transactions on Computers*, Vol. C-25, No. 12, pp. 1226−1241.

I.F. Burns (1974): ''Current software requirements engineering technology'', TRW Systems Group, Huntsville, Alabama.

A.F. Cardenas (1973): ''Evaluation and selection of file organization: A model and system'', *Communications of the ACM*, Vol. 16, No. 9, pp. 540−548.

J.D. Couger (1973): ''Evolution of business system analysis techniques'', *ACM Computing Surveys*, Vol. 5, No. 3, pp. 167−198.

C.G. Davis and C.R. Vick (1977): ''The software development system'', *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, pp. 69−84.

T. DeMarco (1979): *Structured Analysis and System Specification*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ.

M.E. Dickover, C.L. McGowan, and D.T. Ross (1978): ''Software design using SADT'', in *Structured Analysis and Design*, State of the Art Report, J. Hosier (ed.), Infotech, Maidenhead, UK, Vol. 2, pp. 99−114.

M.E. Fagan (1974): ''Design and code inspections and process control in the development of programs'', Technical Report TR-21-572, IBM United Kingdom Laboratories, Hursley Park, UK.

R.M. Fergus (1969): ''Decision tables: what, why, and how'', in *Proceedings of the College and University Machine Records Conference*, The University of Michigan, Ann Arbor, MI, pp. 1−20.

H.J. Genrich, K. Lautenbach, and P.S. Thiagarajan (1980): ''Elements of general net theory'', in *Net Theory and Applications*, W. Brauer (ed.), Lecture Notes in Computer Science, Springer, Berlin, Germany, Vol. 84, pp. 21−163.

K. Grindley (1966): ''Systematics: A non-programming language for designing and specifying commercial systems for computers'', *The Computer Journal*, No. 3, pp. 124−128.

K. Grindley (1975): *Systematics: A New Approach to Systems Analysis*, McGraw-Hill, London, UK.

K. Grindley (1979): ''The role of trigger in Systematics'', in *Formal Models and Practical Tools for Information Systems Design: Proceedings of the IFIP TC-8 Working Conference*, H.-J. Schneider (ed.), Elsevier, Amsterdam, The Netherlands.

C.B. Jones (1979): ''A survey of programming design and specification techniques'', in *Proceedings of the IEEE Conference on Specifications of Reliable Software*, M.V. Zelkowitz (ed.), IEEE Computer Society, New York, NY, pp. 91−103.

B. Langefors (1963): ''Some approaches to the theory of information systems'', *BIT*, Vol. 3, pp. 229−254.

S.S. Lamb, V.G. Leck, L.J. Peters, and G.L. Smith (1978): ''SAMM: A modeling tool for requirements and design specification'', in *Proceedings of the 2nd Annual International Computer Software and Applications Conference* (*COMPSAC '78*), IEEE Computer Society, New York, NY, pp. 48−53.

H.J. Lynch (1969): ''ADS: A technique in systems documentation'', *ACM SIGMIS Database*, Vol. 1, No. 1, pp. 6−18.

J. Martin (1977): *Computer Data-Base Organization*, Prentice Hall, Englewood Cliffs, NJ.

G.A. Miller (1956): ''The magic number seven, plus or minus two: some limits on our capacity for processing information'', *Psychological Review*, Vol. 63, pp. 81−97.

NCR (1969): *A Study Guide for Accurately Defined Systems*, NCR, London, UK.

J.F. Nunamaker, Jr. (1971): ''A methodology for the design and optimization of information processing systems'', in *Proceedings of the May 18−20, 1971 Spring Joint Computer Conference* (*AFIPS '71* (*Spring*)), ACM, New York, NY, pp. 283−293.

J.F. Nunamaker, Jr., B.R. Konsynski, Jr., T. Ho, and C. Singer (1976): ''Computer-aided analysis and design of information system'', *Communications of the ACM*, Vol. 19, No. 12, pp. 674−687.

L. Peters (1978): ''Relating software requirements and design'', *ACM SIGSOFT Software Engineering Notes*, Vol. 3, No. 5, pp. 67−71.

C.V. Ramamoorthy and H.H. So (1977): ''Survey of principles and techniques of software requirements and specification'', in *Software Engineering Techniques*, State of the Art Report, Infotech, Maidenhead, UK, Vol. 2, pp. 265−318.

D.T. Ross (1977): ''Structured analysis (SA): A language for communicating ideas'', *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, pp. 16−34.

D.T. Ross (1980): ''Principles behind the RSA language'', in *Software Engineering*, H. Freeman and P.M. Lewis II (eds.), Academic Press, New York, NY, pp. 159−175.

D.T. Ross and K.E. Schoman: ''Structured analysis for requirements definition'', *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, p. 1977.

M.E. Senko, V.Y. Lum, and P.J. Owens (1968): ''A file organization evaluation model (FOREM)'', in *Information Processing '68: Proceedings of the 1968 IFIP Congress*, Elsevier, Amsterdam, The Netherlands, Vol. 1, pp. 514−519.

D.G. Severance and J.V. Carlis (1977): ''A practical approach to selecting record access paths'', *ACM Computing Surveys*, Vol. 9, No. 4, pp. 259−272.

D.G. Severance and R. Duhne (1976): ''A practitioner's guide to addressing algorithms'', *Communications of the ACM*, Vol. 19, No. 6, pp. 314−325.

D.G. Severance and G.M. Lohman (1976): ''Differential files: Their application to the maintenance of large databases'', *ACM Transactions on Database Systems*, Vol. 1, No. 3, pp. 256−267.

S.A. Stephens and L.L. Tripp (1978): ''Requirements expression and verification aid'', in *Proceedings of the 3rd International Conference on Software Engineering* (*ICSE '78*), IEEE Computer Society, New York, NY, pp. 101−108.

W.P. Stevens, G.J. Myers, and L.L. Constantine (1974): ''Structured design'', *IBM Systems Journal*, Vol. 13, No. 2, pp. 115−139.

D. Teichroew (1970): ''Problem statement languages in MIS'', in *Proceedings of the International Symposium of BIFOA*, Cologne, Germany, pp. 253−270.

D. Teichroew (1971): ''Problem statement analysis: requirements for the problem statement analyser (PSA)'', ISDOS Working Paper 43, The University of Michigan, Ann Arbor, MI, pp. 20−53.

D. Teichroew (1976): ''ISDOS and recent extensions'', in *Proceedings of the Symposium on Computer Software Engineering*, J. Fox (ed.), Microwave Research Institute Symposium Series, Polytechnic Press, Polytechnic Institute of New York, Brooklyn, NY, Vol. 24, pp. 75−82.

D. Teichroew and E.A. Hershey III (1977): ''PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems'', *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, pp. 41−48.

D. Teichroew, P. Macasovic, E.A. Hershey III, and Y. Yamamoto (1980): ''Application of the entity-relationship approach to information processing systems modelling'', in *Entity-Relationship Approach to Systems Analysis and Design: Proceedings of the 1st Conference on Entity-Relationship Approach*, P.P. Chen (ed.), Elsevier, Amsterdam, The Netherlands, pp. 15−39.

S.J. Waters (1972): ''File design fallacies'', *The Computer Journal*, Vol. 15, No. 1, pp. 1−4.

S.J. Waters (1976): ''CAM 01: A precedence analyser'', *The Computer Journal*, Vol. 19, No. 2, pp. 122−126.

S.J. Waters (1977): ''CAM 02: A structured precedence analyser'', *The Computer Journal*, Vol. 20, No. 1, pp. 2−5.

S.J. Waters (1979): ''Towards comprehensive specifications'', *The Computer Journal*, Vol. 22, No. 3, pp. 195−199.

Y. Yamamoto (1981): *An Approach to the Generation of Software Life Cycle Support Systems*, PhD Thesis, The University of Michigan, Michigan.

S.B. Yao (1977): ''An attribute based model for database access cost analysis'', *ACM Transactions on Database Systems*, Vol. 2, No. 1, pp. 45−67.

S.B. Yao and A.G. Merten (1975): ''Selection of file organization using an analytic model'', in *Proceedings of the 1st International Conference on Very Large Databases* (*VLDB '75*), IEEE Computer Society, New York, NY, pp. 255−267.

E. Yourdon and L.L. Constantine (1979): *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ.

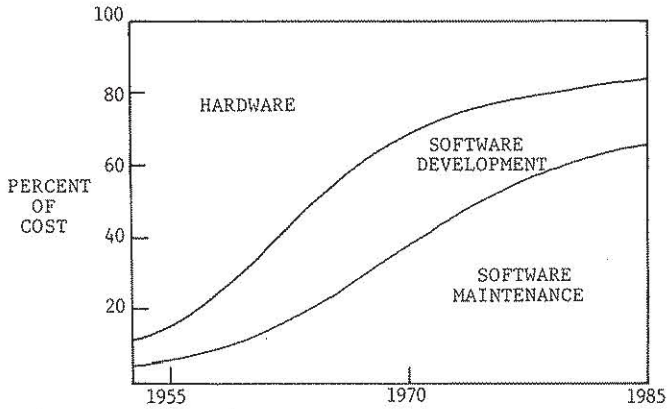



Figure 1. Hardware-software cost trends.

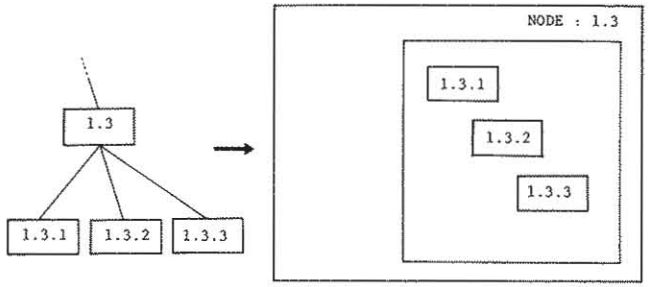



Figure 2. Relationship of the labelled tree and activity diagrams.

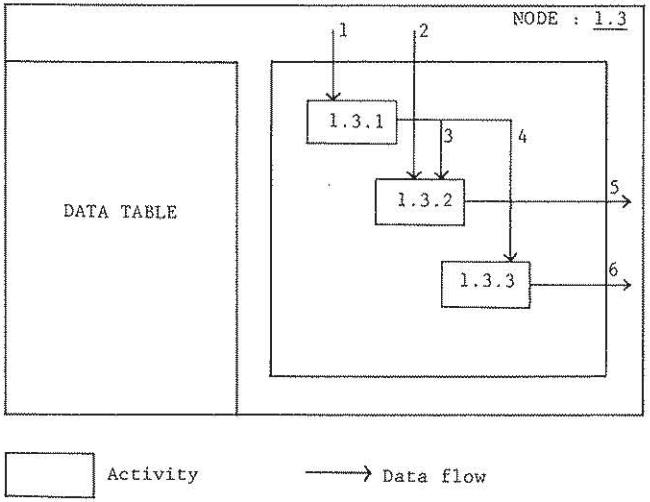



**Figure 3. An activity diagram.**

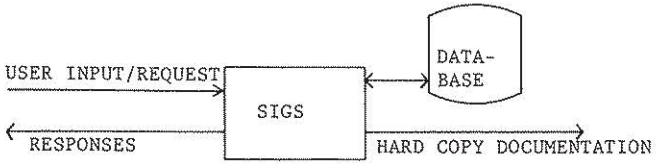



**Figure 4. An overview of SIGS.**

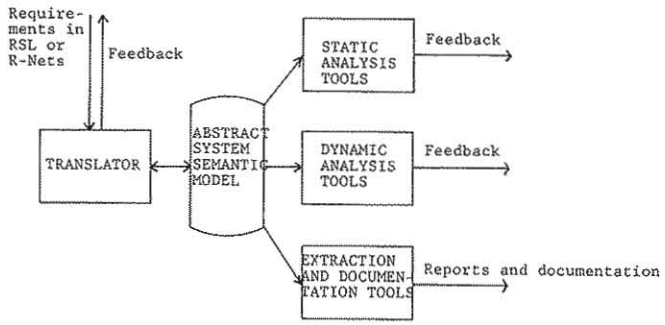



Figure 5. An overview of SREM.

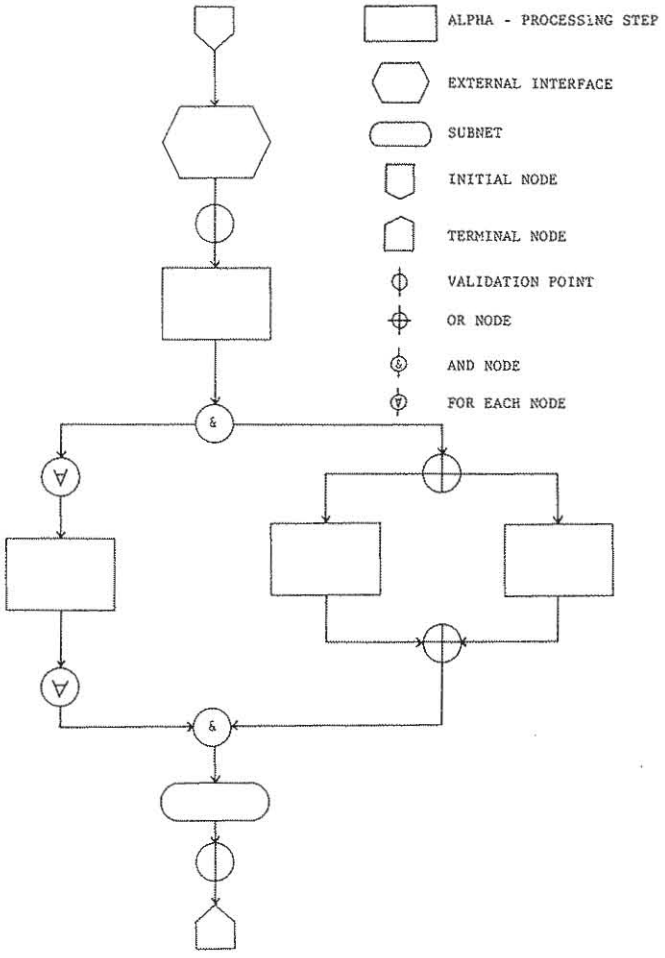
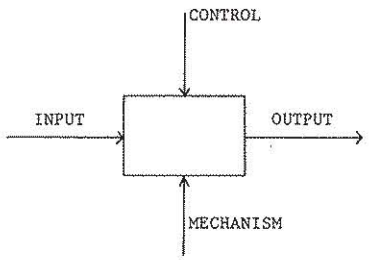



Figure 6. A Requirement Net.




Interpretation: The box is a valid transformation of the input into the specified output provided the support mechanism is available and the correct control is applied.

Figure 7. SADT fundamental building block.

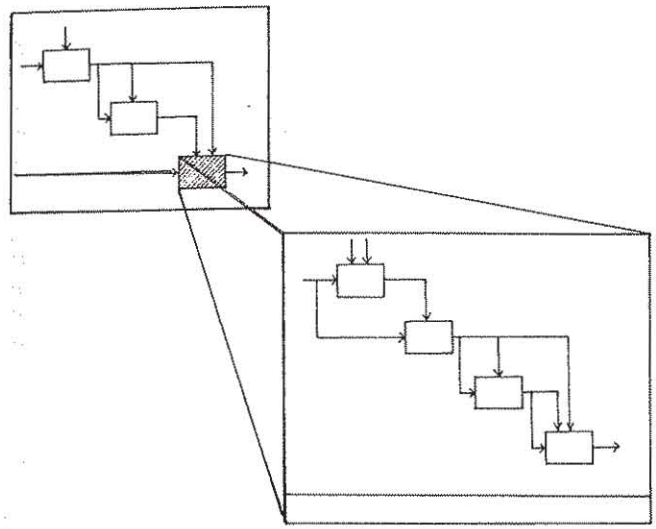


Figure 8. SADT diagram and the decomposition/detailing of a box into a separate diagram.

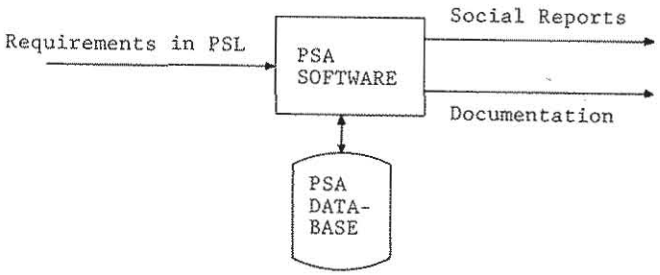



Figure 9. An overview of PSL/PSA.

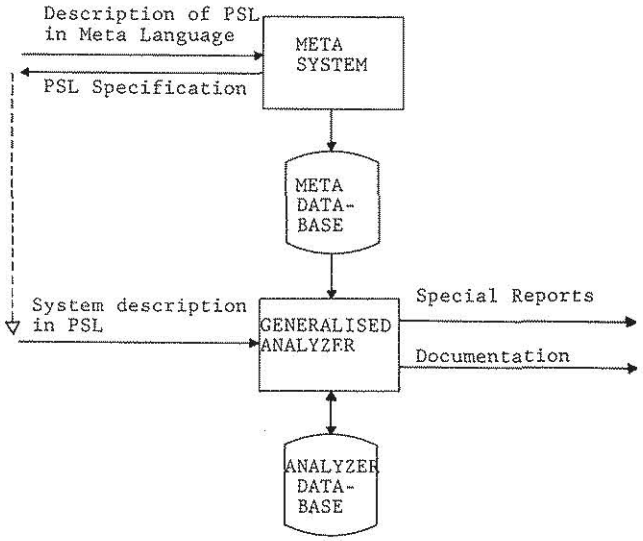



Figure 10. Application of the META/GA system for generating specific PSL/PSA system.

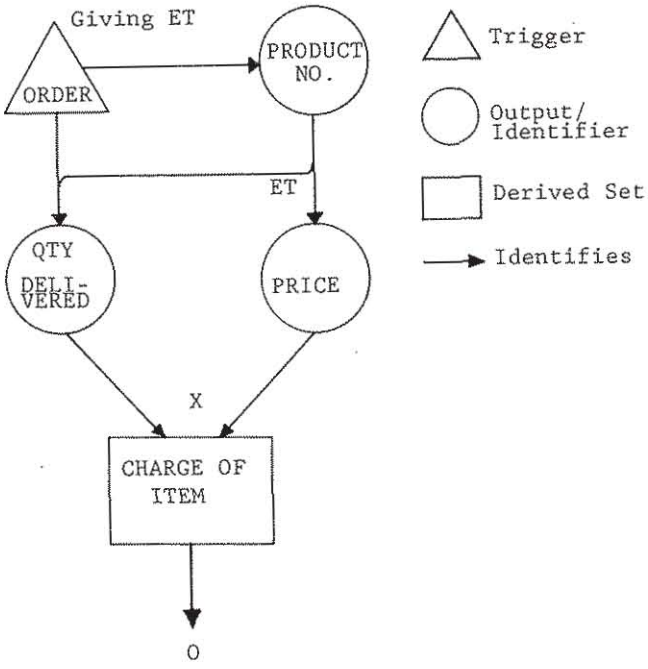



Figure 11. A systematics flowchart.

Table 1. Comparison of defect detection methods
(from Jones, 1979)

| | Design Review | Machine Testing | Correctness Proofs | Models or Prototypes |
|---|---|---|---|---|
| Omitted Functions | Good | Fair | Poor | Good |
| Added Functions | Good | Poor | Poor | Fair |
| Structural Problems | Good | Fair | Poor | Fair |
| Algorithm Problems | Fair | Fair | Good | Good |
| Human-factor Problems | Fair | Poor | Poor | Good |

Table 2. Summary of findings

| | SAMM | SREM | SADT | ADS/ SODA | PSL/ PSA | Sys- tem- atics |
|---|---|---|---|---|---|---|
| Validation | | | | | | |
| – Completeness | | | | | | |
| – Continuity | ✓ | ✓ | ✓* | ✓ | ✓ | ✓* |
| – Consistency | ✓ | ✓ | | ✓ | ✓ | |
| – Redundancy | ✓ | | | | | |
| User Verification | | | | | | |
| – Use of Natural Language | | | | | | |
| – Generation of Documentation | ✓ | ✓ | | ✓ | ✓ | |
| – Graphic Aid | ✓ | ✓ | ✓ | | ✓ | ✓ |
| – Bounded Rationality | ✓ | | ✓ | | | ✓ |
| – Simulation/ Prototype System | | ✓ | | | | |
| Optimisation | | | | | | |
| – File | | | | ✓ | | |
| – Process | | | | ✓ | | |
| Maintenance | | | | | | |

* Manual