A Language for provably-complete System Specifications

Kit Grindley, John St.Quinton, Frances Stubbs
The Systematics Research Group
miexp@yahoo.co.uk

Abstract - *Hitherto, the automated generation of computer–based systems has been impeded by the lack of a suitable specification language. Specifications written in such a language should demonstrate three essential characteristics: be independent of programming strategy; be provably-complete; account for variety of output in terms of the variety of previously supplied input, the variety of each individual output being related to its stimulus. This paper proposes a theory of information-based systems, "Systematics", and a language derived from it, the "Systematics Specification Language"(SSL); a language that demonstrates these three characteristics. A Systematics specification serves as a program for implementing an information system on a Systematics Engine - an abstract engine having infinite memory and instantaneous processing capability. Storage, deletion and retrieval strategies enabling finite computers to employ the minimum database needed to satisfy the requirements, can be automatically deduced from Systematics specifications. The significant implications of Systematics to the integrity of safety critical systems are discussed.*

Keywords: System Specification, Automated Programming, Database Design, Safety Critical Systems.

1 Background

Information System (IS) development has been constrained by the practical expedient of assuming a fixed and unerring system requirement. This fallacy is succinctly expressed by "The Fixed Point Theorem" [8] *There exists some point in time when everyone involved in the system knows what they want and agrees with everyone else.* Practical experience demonstrates this assumption to be false: so why do we develop an IS as a one-time project and not as an evolving requirement? One early clue [12]: *"After 20 years, writing a lot of programs, reading a lot more, and debugging even more of them, my overall impression of our business is that we are struggling, with our unaided minds, on something far too big for us!"* It is now recognised that human beings cannot continually evolve the complex and interrelated structural elements that underpin computer programs. Twenty years ago, Price Waterhouse reported a backlog of one and a half million programmer years [9]. It would seem we have no alternative but to pretend the Fixed Point Theorem is true. Software development now relies predominantly upon standard programs or 'packages' [4]. However, software packages actually create a new evolutionary requirement, the pursuit of technological progress, yet fail to address the most pressing need: evolving IS requirements.

2 The characteristics of an evolving Requirements Specification

The automation of programming has been advocated as the means of continually adapting IS. Many program generators were marketed in the 1980's as Computer Aided Systems Engineering 'CASE' tools [10]. A computer has the potential to manipulate existing programs, however complex, and generate amended versions to meet changed requirements. However, a suitable statement of requirements has to be fed to such a program generator. A 'suitable' specification will include three essential characteristics:

Freedom from Program Strategy A requirements specification should be free from program strategy. Once programs are manipulated automatically, the problem of continually evolving them changes from one of 'changing the programs' to one of 'changing the requirement specification'. The requirement specification must be free from considering the effect of requirement change on consequent changes to programs and program strategy. The point was recognised and expressed by Bosak et al [2] … *the problem definition is buried in the … algorithmic statement of the solution ...*

Completeness Unless the program generator is to guess the users' needs, a complete requirement statement must be provided. Completion can only be reliably achieved if it is provable.

Variety Information systems produce outputs that differ from each other. A requirements specification should account for the output variety in terms of the input variety supplied.

CASE tools have not achieved their objective of automating the design and production of programs so as to satisfy a given requirement. It is suggested that the reason for the failure in previous theories of data processing is

that they are unable to produce requirements specifications with these three essential characteristics.

3 Systematics engines

Systematics was proposed by Grindley as a language for describing IS requirements independently of implementation strategy [5], [7]. Stimulus-related output specifications were suggested as the basis for such a language [6]. The use of Systematics as a query language has already been demonstrated by Sernadas [11].

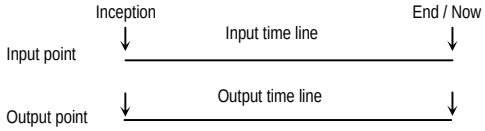
The Theory of Systematics, in conjunction with its associated system specification language SSL, has subsequently been developed by the current authors to the point where any IS requirement, written in SSL, satisfies a notional Systematics engine: an engine having infinite storage capacity and instantaneous processing capability; an engine in which all inputs are stored, in their order of reception, forever; an engine in which nothing else is stored. Systematics engines have one input channel and one output channel, both of which can be sequentially multiplexed. As will be described later in 'Database Design', these subsequent developments also lay the foundation for the automated generation of programs that verifiably match their requirement specification.

The Systematics engine concept affords the opportunity to produce a requirement specification that conforms to the suitability tests described, being unencumbered by the constraints imposed by real computers. In particular: primary keys are not required for the identification of single items of data since each item has a unique input or output (I/O) time; data summaries and the results of calculations do not need to be stored since they can be recalculated from the original input; output requirements may be specified in terms of inputs, since the original inputs are stored forever and are always available as output components.

4 Systematics Theory of Information Structure

4.1 System-time

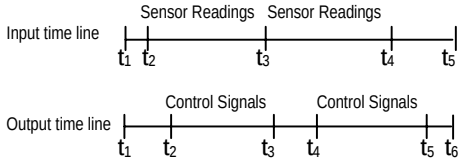
IS components are divisions of 'System-time', the subset of time which exists at the input point and the output point of a Systematics system, and from its inception until its end, or, if continuing, until now.



The input time line and the output time line are each a one-dimensional continuum of what is termed 'System-time'.
In a current system, as System-time passes, the input and output points repeat to the right of the diagram, and the time lines increase in length.

Figure 1. System-time

I/O time lines are divided into separate Periods. Separate Periods which have a recognisable type form the Inputs and Outputs of the system. Periods with no recognisable type are of no significance.



In a control system for a nuclear reactor for example (see Section: Systematics Specification Language), the inputs are readings from the reactor's sensors.
The outputs are signals which control the reactor's operation.
On the input time line, Periods t_2 to t_3, and t_3 to t_4 are Inputs of the type Sensor Readings. The Periods t_1 to t_2 and t_4 to t_5 have no type, and are not Inputs.
On the output time line, Periods t_2 to t_3, and t_4 to t_5 are Outputs of the type Control Signals. The Periods t_1 to t_2, t_3 to t_4 and t_5 to t_6 have no type, and are not Outputs.
t_1 to t_2 on the input time line is a Period of input System-time and is thus different from the Period t_1 to t_2 on the output time line which is a Period of output System-time.

Figure 2. Priods of System-time

4.2 Resolution

All Periods of System-time can be resolved into sets of smaller Periods, and Instants of System-time. Instants are the smallest recognisable divisions of System-time, and form the limits of I/O resolution.

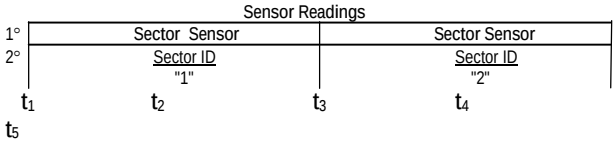
Periods are termed the parent of the Period sets and Instants they contain at the next 'degree of resolution'. Parents contain none, one, or many Period sets. A Period set consists of one or more Periods of the same type. Parents contain none, one, or many Instants of different types.

Period sets and Instants cannot be of the same type as their siblings, or any ancestor.

Instants have a value, which may be null. Periods do not have a value.

Inputs and Outputs are separate Periods – that is, they have no parent.

The size of an Input or Output is measured by the number of Periods in its resolution.



The Periods and Instants are labelled with their type. Period types are in bold. Periods have no value. Instant types are underlined. Each Instant has a value.
The Input of type Sensor Readings is a Period of System-time, stretching from t_1 to t_5. It has no parent. At its first degree of resolution, the Sensor Readings Period contains a set of two Periods of type Sector Sensor stretching from t_1 to t_3 and t_3 to t_5 respectively.
At the second degree of resolution Sector Sensor t_1 to t_3 contains one Instant of System-time t_2 of type Sector ID with value "1" And similarly for Sector Sensor t_3 to t_5. Only Instants are discovered at the second degree of resolution, and thus no further resolution is possible.

Figure 3. Resolution

5 Systematics Theory of Requirement Specification

5.1 Identification

Period sets and Instant sets are identified by their type. Each Period and each Instant has a natural System-time order by which it can be identified. Systematics engines have only one input and one output channel. Periods and Instants are thus uniquely identified by their type and their input or output System-time specified with reference to a known System-time, normally that of a Stimulus e.g.:

The last **Sensor Readings**
(i.e. the last before the current Stimulus)

5.2 Union-Continuum

Periods and Instants have the property of belonging to a resolution. Instants also have the property of value. It follows that a closed path exists between Periods or Instants with specified related properties. Such closed paths are termed "joins". Any component participating in a join may itself be joined to other components. Each join is a closed path, therefore the combination of the two joins is also a closed path. Similarly, a third, and any number of joins may be added, and therefore, by induction, the resulting chain of joins will always be a closed path. A chain of joins identifies a Correspondent set or an Origin, and every branch of the chain ends with a reference to the stimulus. When each Correspondent set or Origin of the same type is referenced to the stimulus by a necessarily different path (see Size 5.3.3 and Value 5.3.4 and also Variety 7 below) it is termed a Union Continuum (UC). E.g. :

The **Sector Sensor** contained in the **Sensor Readings** which is the current Stimulus

5.3 Variety

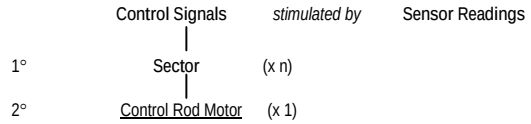
Inputs and Outputs (I/Os) exhibit four kinds of variety: Type. The I/Os are of different types, e.g. **Sensor Readings**, **Control Signals**, etc. ; Timing. I/Os are input and output at different times; Size. I/Os of the same type may contain a different number of each component type, e.g. different **Sensor Readings** Inputs may contain different numbers of **Sector Sensor** Periods; Values. I/Os of the same type may contain different values e.g. the value of Sector ID may be either "1", "2", etc.

The four kinds of variety, Type, Time, Size and Value, are specified as follows:

5.3.1 Variety of Type

Every Output type has a specified Stimulus type, and a specified Resolution Format. The first degree Resolution Format shows the Instant types and Period set types produced by the resolution of the Output.

The second degree Resolution Format shows the Instant types and Period set types produced by the resolution of each Period set type produced by the first degree resolution. And so on.

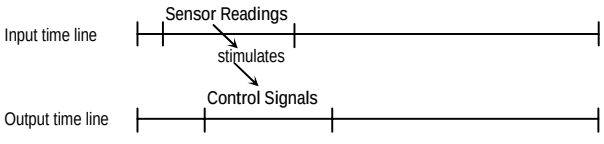


A Control Signals Output is produced for every Sensor Readings Stimulus.
At the first degree of resolution of the Output Control Signals, a set of many Periods of type Sector is produced.
At the second degree, one Control Rod Motor Instant is produced for each Sector Period. Since all these components are Instants no third degree resolution is possible.

Figure 4. Variety of Type

5.3.2 Variety of Timing

Every type of Output is stimulated by one or more specified types of Input, termed the 'Stimuli' of the Output types. The order of Outputs of the same type is a repetition of the order of their Stimuli.



One Output of the type Control Signals in produced when an Input of type Sensor Readings (specified as its Stimulus type) is received.

Figure 5. Variety of timing

5.3.3 Variety of Size

The size of a stimulated Output corresponds to the size of a specifically assembled Stimulus Reference

Structure, consisting of sets of previously input Periods, which maps onto the Output. Each input Period set corresponds to a stimulated output Period set and is termed the output set's 'Correspondent Set'. Each input Period corresponds to a stimulated output Period and is termed the output Period's 'Correspondent'.

Each Stimulus Reference Structure is headed by the Stimulus of its associated Output. First-degree Correspondent Sets are specified by a UC with reference to the Stimulus. Second-degree Correspondent Sets are specified by a UC, with reference to their first-degree Correspondent. And so on. Every Correspondent Set is thus ultimately referenced to the Stimulus.

A set size specification is given for each Period type in the Output Resolution Format.

Output Resolution Format	Set Size Specification	Stimulus Reference Structure		Associated Output Periods
Control Signals	The Stimulus	Sensor Readings	→	Control Signals
1° Sector	The Sector Sensor(s) contained in the Stimulus	Sector Sensor (t_1 to t_3)	→	Sector (1)
		Sector Sensor (t_3 to t_5)	→	Sector (2)

The above example continues the specification of the Output Control Signals whose Resolution Format, Stimulus type and timing were specified in Figures 4 and 5, and for the Sensor Readings shown in Figure 3.

The first set size specification is always "The Stimulus". In the above example this specification produces a Sensor Readings set of one Correspondent.

The specification of each first-degree Correspondent Set always produces a single set. First-degree specifications always refer to the Stimulus. In this case there is one Set Size Specification, selecting the Sector Sensor Periods contained in the Stimulus. Two Sector Sensor Periods are selected.

The specification of second-degree Correspondents always produces one set for each first-degree parent Correspondent Period, and always refers to the first-degree parent Correspondent. In this example there are no second-degree Period sets.

When the Stimulus Reference Structure is assembled, it is mapped onto its associated output Periods, thus determining the Output's size. Since one Correspondent Sensor Readings input Period is selected, one Control Signals output Period is produced. Since two Correspondent Sector Sensor input Periods are selected, two Sector output Periods are produced.

Figure 6. Variety of Size

5.3.4 Variety of Value

The value of each output Instant is a specified function of the value of one or more specified input Origins. Since each discovered Period in a resolution contains a single Instant of any given type, the specification must result in a single value. Origins are previously input Instants. Each Instant type in an Output Resolution Format has its Origin specified by a UC with reference to its parent Period's Correspondent, and thus ultimately to the Stimulus.

Output Resolution Format	Value Specification
Control Signals	
1° Sector	
2° Control Rod Motor	= function of Temperature contained in parent 's Correspondent.

The value of Control Rod Motor is specified as a function of the value of an Origin. The Origin is specified as the Temperature Instant in the parent Sector's Correspondent. Since the Sector Correspondent was specified with reference to the Stimulus (see Figure 6), the value of Control Rod Motor is also specified with reference to the Stimulus.

Figure 7. Variety of Value

6 Systematics Specification Language

For brevity, the Systematics Specification Language (SSL) will be described by means of a worked example: specifically an SSL specification for a program to control a nuclear reactor.

In the reactor core, uranium atoms decay, releasing neutrons which bombard other uranium atoms, causing them in turn to decay, thus setting up a chain reaction. This process releases energy, which heats the core. The temperature of the core is monitored; if it becomes too high, boron control rods are lowered into the core, to absorb neutrons and thus slow down the chain reaction. The core is divided into sectors. Each control rod is raised or lowered according to the temperature of that sector, and the current position of the rod. If the temperature reaches a danger level, all the control rods are released, and drop under gravity into the core, stopping the chain reaction and closing down the reactor completely; the 'Emergency Shutdown'.

In this simplified example, the system produces two outputs: Control Signals, sending control signals to control rod motors in response to sensor readings from the reactor, and Management Report, which periodically reports statistical data from the reactor.

The Systematics Specification Language (see Figure 8) consists of a graphic syntax giving a 'picture' of the Outputs, combined with a *Union-Continuum* which adopts a linear syntax to specify each output element.

	Period / Instant		**Correspondent** / Origin	**Union-Continuum**
1.	**Control Signals**		**Sensor Readings**	Š
2.	Emergency Shutdown		f_1(Temperature)	\ Ç$_{\tau 1}$
3.	1 **Sector**		**Sector Sensor**	\ Ç
4.	Sector ID		Sector ID	\ Ç
5.	1.1 **Control Rod**		**Control Rod Sensor**	\ Ç
6.	Control Rod ID		Control Rod ID	\ Ç
7.	Control Rod Motor		f_2(Temperature, Control Rod Position)	\ Ç$_{\tau 1}$ \ Ç

Figure 8 (a). SSL Specification for Nuclear Reactor Control System. (Control Signals Output)

	Period / Instant	Correspondent / Origin	Union-Continuum
8.	**Management Report**	**Period End**	Š
9.	Date	Date	\ Ç
10. 1	**Sector**	**Sector Sensor**	\ **Sensor Readings** *[last before Š]* / *different* Sector ID
11.	Sector ID	Sector ID	\ Ç
12.	AvTemp	Average(Temperature)	\ **Sector Sensor** \ **Sensor Readings** *[since previous Š]* / Sector ID = Sector ID \ Ç
13.	OptTemp	Optimum Temperature	**\ Sector Data** *[last before Š]* / Sector ID = Sector ID \ Ç

Figure 8 (b). SSL Specification for Nuclear Reactor Control System (Management Report Output)

6.1 Period/Instant column

The first column shows, by indentation, the Resolution Format of Outputs of a given type.

An index number shows the resolution level: 1, 1.1 etc. Period Types are in bold; Instant types are underlined.

In Figure 8, each Output of the type **Control Signals** contains an Instant of type Emergency Shutdown (line 2) and, at the first level of resolution (1), a set of Periods of type **Sector** (line 3). Each **Sector** Period contains an Instant of type Sector ID (line 4) and, at the second level of resolution, (1.1), a set of Periods of type **Control Rod** (line 5). And so on.

6.2 Correspondent/Origin Column

The second column shows the Correspondent type for each output Period type.

The Correspondent of the Output is always the Stimulus.

In Figure 8, every **Control Signals** Output is stimulated by a **Sensor Readings** Input (line 1). At the first-degree of resolution, the Correspondents of **Sector** Periods are Periods of type **Sector Sensor** (line 3). At the second-degree of resolution, the Correspondents are Periods of type **Control Rod Sensor** (line 5).

The second column also shows each Instant type in the Output hierarchy as having a value equal to a function of one or more input Instant types, termed its 'Origins'.

In Figure 8, the value of a Sector ID output Instant is equal to the value of an input Origin of type Sector ID (line 4). The value of an Emergency Shutdown output Instant (line 2) is a function, f_1, of input Instant Origins of type Temperature. (The nature of the function, f_1, is not shown here, although of course it would have to be specified.)

6.3 Union-Continuum Column

The third column shows the Union-Continuum that identifies each Correspondent set and each Origin. At the lowest level, the Correspondent *is* the Stimulus Š. At each degree of resolution the UCs link Correspondent sets and Origins to the parent Correspondent, Ç, and thus, ultimately, to the Stimulus.

Periods may be identified by their System-time, shown in italics in square brackets, *[ ]* .

Sensor Readings *[last before Š]*

in Figure 8, line 10, identifies the last **Sensor Readings** Period input before the System-time of the Stimulus.

Periods may be identified by the value of the Instants they contain. "/" means "containing".

Sector Data *[last before Š]* / Sector ID =

in Figure 8, line 13, identifies the last Period of the type **Sector Data** containing an Instant of the type Sector ID with a particular value.

Instants are identified by the identified Periods which contain them. "\" means "contained in".

Temperature \ **Sector Sensor**

in Figure 8, line 12, identifies the single Instants of the type Temperature contained in each of the identified Periods of the type **Sector Sensor**.

Unions continue until the Stimulus, Š, is reached, - hence, 'Union-Continuum'.

Sensor Readings Š

in Figure 8, line 1, identifies the **Sensor Readings** Period which is the Stimulus itself.

UCs may refer to the Correspondent, Ç, which in turn refers to the Stimulus.

Sector ID \ Ç

in Figure 8, line 4, identifies the Sector ID Instant contained in the Correspondent. The Correspondent, in this case, is a **Sector Sensor** Period, which, in turn, is specified as contained in its own parent Correspondent, which is the Stimulus.

UCs may refer to Correspondents at different levels of resolution, whose Correspondents in turn have UCs linking them to the Stimulus.

Temperature \ Ç₁

in Figure 8, line 2, identifies Instants of the type Temperature contained in the Correspondents at level 1. These are **Sector Sensor** Periods (line 3).

Union-Continua may occupy more than one physical line, and may employ parentheses to avoid ambiguity.

7 Systematics Specifications Satisfy the Essential Characteristics

Three characteristics of a requirements specification from which programs can be generated automatically have been identified. Systematics theory enables specifications having these characteristics to be written.

Freedom from Program Strategy A Systematics requirement specification is free from program strategy – most noticeably: inputs and files are not specified. No use is made of primary keys. No purging strategies or summaries of inputs are proposed to minimise the size of the database.

Completeness Completeness in a Systematics system is defined as meaning that every variable aspect of an Output: type, size, System-time and value, is determined from variable aspects of stimulus-related Inputs. It has been shown by induction that: Every variable aspect of an Output is linked, by a closed path 'Union- Continuum' to its Stimulus. It follows that the automatic, or manual, navigation of the specification, from output to stimulus, can test for closure.

Variety Output type, and the Period types and Instant types in the resolution of periods of that type, are specified for Stimulus type (see Figure 8). The System-time of an Output is determined by the System-time of its Stimulus (see Figure 5). The Requisite Variety (RV) of size is that the sizes of output period sets of the same set-type are potentially different. This is ensured by union-continua which select each correspondent period set with reference to the correspondent of its output's necessarily different parent (see Figure 6). Similarly, the RV of value is that the values of output instants of the same type are potentially different. This is ensured by specifying the selection of each origin with reference to the correspondent of its output's necessarily different parent (see Figure 7). The germ of variety in the Output's Stimulus is thus differentiated throughout the Output structure, limited only by the variety contained in all the Inputs ever received.

A reformulation of Ashby's Law of Requisite Variety [1] expresses the basis of Systematics.

The Law of Requisite Variety – Systematics form
The larger the variety of continuum-located inputs available to an information system, the larger the variety of outputs it is able to produce.

8 Database Design

Since a Systematics specification describes Outputs in terms of Inputs, the inputs, which form the infinite database of a Systematics engine, can be deduced from it. This infinite database can be reduced to a practical size by the automatic deduction of storage and purging rules from a Systematics specification.

8.1 Removing unnecessary Periods and Instants

Periods and Instants required only to produce the Output they stimulate need not be stored.

Control Rod Position \ Ç

Control Rod Position (Figure 8, line 7) contained in the Correspondent **Control Rod Sensor** Period, which in turn is contained in its parent Correspondent **Sector Sensor** Period, which is contained in the stimulating **Sensor Readings** Period, is referred to nowhere else in the specification, and so need not be stored.

Function parameters need not be stored if an updated result is stored each time a parameter is input.

AvTemp = Average(Temperature) \ **Sector Sensor** \ **Sensor Readings** / Sector ID =

The number of Temperature Instants, and the sum of their values, can be stored, together with the identifying Sector ID Instant, each time a **Sector Sensor** \ **Sensor Readings** Period is input. The **Sector Sensor** Periods need not be stored. (Figure 8, line 12)

If the last Period containing each value involved in a specified join is the only one required, only that one need be stored.

Optimum Temperature **\ Sector Data** *[last before Š]* / Sector ID

Only the last Optimum Temperature **\ Sector Data,** for each different value of Sector ID, needs to be stored. (Figure 8, line 13)

8.2 Removing unnecessary Correspondents

Specified Correspondent Periods may be removed when their selection is no longer possible.

Sensor Readings *[since previous Š]*

Since **Sensor Readings** Periods prior to the previous **Period End** Stimulus are never referred to, they can be removed when a new **Period End** Stimulus is received. (Figure 8, line 12)

The instantaneous search capability of the abstract Systematics engine can be compensated by indexing stored Periods by the Instants involved in a specified join. In this way, primary keys are deduced for Periods containing Origins, and secondary keys are deduced for Periods selected as Correspondents.

9 Applications of Systematics

SSL overcomes a major problem associated with current methodological and programming techniques used for the development of Safety Critical Systems - Design Revisions. Budgetary constraints and delivery deadlines can erode the full integrity of an evolving system specification. With SSL, the regeneration of a modified specification invokes the same completeness criterion that was applied to the original proven version: the Union-Continua will either be complete, or incomplete.

SSL does not involve Expert System rules nor is it dependant on the interpretation of an arbitrarily complex text-based methodological procedure such as SSADM [3]. SSL is based on formal logical completeness: the completeness of the union of data contained within information sets existing between Output and Input(s). Manual or Auto-Navigation between the data requirement for each specified Output and the associated specified Input(s), highlights: redundant inputs, data inadequacies and ill-supplied derivatives.

An SSL specification for a system can either be written in its entirety or as a skeletal functional specification for an SSL system-generator incorporating auto-navigating Union-Continuum search procedures. Any functional system can be specified by SSL, including 1st and 2nd order Cybernetic systems. Creating compound interactive systems involves no more than the iterative application of SSL and O/I matching.

Fundamentally, SSL specifications describe requirements assuming they are to be satisfied by a Systematics Engine with infinite storage, and where all Inputs are remembered forever. The identification methods proposed in SSL for such infinite time-series enable rules to be applied to the specification which reduce the stored information to the minimum required to satisfy the output requirement. This enables the automatic design and revision of application databases, which is the only remaining barrier to the automatic generation of computer programs.

Automated database design, and the implementation of SSL as an automated system-generation tool are the current tasks of the Systematics Research Group.

References

[1] Ashby, W.R (1956) *An Introduction to Cybernetics*, Chapman and Hall

[2] Bosak, R. et al (1962) *An Information Algebra*, Comm ACM. Vol *5*. No 4

[3] Clare, P. Coe, I. Downs, E. (1992) *Structured System and Design Method*, Prentice Hall

[4] Compass (1998) *Proportion of Systems Represented by Packages*, Survey by Compass.

[5] Grindley, C (1966) *Systematics - a Non-programming Language for Designing and Specifying Commercial Systems for Computers*, Computer Journal, Vol 9, pp 124 - 128

[6] Grindley C., Stevens, W (1972) *Principles of the Identification of Information*, Proc. FILE '68

[7] Grindley, C (1975) *Systematics - A New Approach to Systems Analysis*, McGraw-Hill

[8] Paul, R.J (1994) *Why Users Cannot 'Get What They Want'*, International Journal of Systems Design, Vol 1, No.4.

[9] PW (1986) *Major Issues*, Information Technology Review (pub Price Waterhouse)

[10] PW (1989) *CASE for improvement*, Information Technology Review (pub Price Waterhouse)

[11] Sernadas, A (1981) *Systematics: Its Syntax and Semantics as a Query Language (1) and (2),* Computer Journal Vol 24, (No 1 February, No 2 May)

[12] Weinberg, G (1971) *The Psychology of Programming*, Van Nostrand Reinhold