

# SOFTWARE LIFE CYCLE METRICS: FROM PROJECT INITIATION THROUGH DESIGN

Dr. Carl A. Singer 8\*747-5194  
Advanced Software Engineer  
Data Systems Resource Management  
P.O. Box 8555 - C7-7621  
Philadelphia, PA 19101

## Abstract

The measurement of target system complexity, performance and quality is vital throughout the software life cycle. This is especially true during a system's formative phases: requirements specification and design. The natural tendency to emphasize measurement only where adequate metrics and data gathering capabilities exist however, neglects significant opportunities. Measurement beginning with coding and continuing through the remaining life cycle phases (testing, implementation and maintenance) neglects true target system complexity. Specifically, a complex solution to a simple process is misinterpreted. The resultant system, not the target system, becomes the baseline for metrics. This paper introduces software metrics as an integral part of the requirements specification design phases of the software life cycle.

"There is nothing as inevitable as a mistake whose time has come."

.... Murphy, et al.

## A Question Of Focus

Most metrics focus on quantifiable and readily measurable elements. Those elements that have traditionally been easy to measure, such as lines of code or data base size, have been emphasized. Metrics for more obscure elements, such as system complexity, have, for the most part, been neglected. Also, metrics measuring project size, and those measuring system size are erroneously intermingled.

How large is a system? The answer is seldom clear, and it varies along various stages of the system life cycle. There are various measures depicting the resultant system. These include lines of code, number of messages, number of interfaces, etc. There are measures depicting the size of intermediate deliverables such as the preliminary design specification, the PDL or the detailed design specification, etc. There are level of effort measures such as man-years. There are resource measures that capture cost and/or value such as project cost, total budget, etc. Finally, there are input measures depicting the initial requirements statement. These may include requirements document size (words or paragraphs) or number of requirements (number of "shalls" in the document).

Measuring the resultant system against the requested system is the ultimate aim of requirements and design metrics. Metrics must establish a suitable baseline to accomplish these measurements. Beginning with the initial requirements statement, each stage of the systems life cycle should be measurable and should serve as a baseline for measuring performance of the next stage.

## Using The Wrong Baseline

Performance and efficiency metrics often measure against a baseline that reflects the designed target system. While this may prove useful for measuring efficiency within the coding and implementation efforts, it ignores measurement of the design process. The "as requested" system not the "as built" system should be the fundamental baseline. Specifically a complicated or poorly designed "solution" is lost to the metrics and quality assurance data bases.

The nature of systems development is such that we seldom, if ever, have the opportunity to compare multiple attempts to meet the same requirements. It is fruitless, therefore, to attempt to compare disparate systems - i.e., "Johnny built his project management system in six months and Suzie took only four months to build her project management system."

Consider this simple problem, "write a program to sort 100 names into alphabetical order". Various designs and algorithms will result in different resultant systems. Even given the same algorithm and the same language, one person may implement the algorithm using 50 lines of code, another using 25. If one person took three days to produce 50 lines of tested code (16.7 lines / day), the other only two days to produce the 25 lines of tested code (12.5 lines / day). Which was a better design? Which was better implementation? Who did a better job and by what measures?

We wish to measure requirements because this provides metrics for a true system baseline. Given a good requirements-based metrics baseline, we can better evaluate both design and implementation. Many requirements metrics are unsatisfactory however, because they are inconsistent in comparing one project or sub-project with another. Document size for example, may reflect format and style (terseness, verbosity, margins); "shalls" may not properly capture compound sentences nor the complexity of the underlying requirements. Using a rigorous Requirements Statement Language (RSL), coupled with clear guidelines, may provide more useful measures.

## Requirements Statement Languages

Requirements Statement Languages (RSL's) also known as problem statement languages were introduced in the mid-1960's to formally capture requirements for an information processing system. Early efforts targeted toward management information processing systems, as opposed to real-time systems. The RSL's provided input to a comprehensive systems design methodology that includes module and data base design. Examples include PSL/PSA, ADS, SADT and SREM.

Despite the availability of RSL's, contractors usually state requirements using narrative. A system concept is described and the narrative continues to depict a general solution scheme. In describing real-time systems, the requirements statement should identify all objects (physical objects, external systems, human interfaces) that act within the system as well as verbs (the action or transactions that transpire among the objects). From this skeletal description the requirements can be reviewed in terms of interfaces (any time an object communicates with another object) and messages (that which is sent across an interface). This process is not necessarily straightforward because both objects and verbs are often imbedded in the narrative and may be stated ambiguously.

The Key To Good Metrics Is Consistency Across Samples.

A Requirements Statement Language (RSL) may be used to translate the narrative system description into a more useful form. The RSL should describe the required system in a non-procedural manner. Essentially, non-procedural means saying "what" should the system do, NOT "how" should the system do it. Note that constraints may be procedural (e.g., "safety check must be completed using approved algorithm ABC"). The requirements are described using objects (things which do, or are done to) and verbs (actions or transactions). A procedural statement might state: "Read all balloon messages, then search for those reporting for the second consecutive time, then "activate" those balloons ...." A non-procedural requirements statement might be: "Balloons shall be activated when two consecutive messages are received." The non-procedural statement does not pre-select between the design alternatives of "read all messages (first) then process them (as a batch)" and "read and process all messages (serially)".

The goal is to produce a consistent baseline that reflects the system "as required". Ideally, anyone, using any methodology, should produce a requirements statement consistent with that produced by anyone else. Consistency is achieved by minimizing the need for decisions or choices, either through procedures and rules or by eliminating, as much as possible, the underlying need to make choices during the requirements statement process.

A formal description of the required system is built by searching through the narrative requirements document . First, the narrative is reviewed for nouns and verbs (objects and actions). Note: the flexible (inconsistent) use of the English language (such as "shall" and "will") precludes simple counting. Consider the following Customer Requirements Document:

Customer Requirements Document - for PROJECT WINDBAG

- 1.0 WINDBAG is a real-time weather balloon information monitoring system. This project shall develop a data processing facility (DPF) that shall communicate with weather balloons as they float across the sky.
- 2.0 The weather balloons automatically transmit a coded message each minute. The transmission consists of: balloon ID code, current weather information and balloon position data.
- 3.0 When the DPF has received two consecutive transmissions from a specific balloon, it declares the balloon to be "active" and monitors weather information at 10KM position intervals.
- 4.0 The Central Processing Control Point (CPCP) is to be notified of all confirmed balloon contacts as well as of all lost contacts. On request from the CPCP, the DPF will provide all information on a particular balloon and shall then purge that current information. On request from the CPCP, the DPF shall provide current information on all of its balloons (without purging that information).
- 5.0 An operator working through the operator console shall be able to request reports for either a single balloon or for all balloons. The operator may also request that data be purged from the DPF. The operator may instruct the CPCP to do a complete data dump.

The objects in the required system are:

1. the weather balloons (BALLOON)
2. the data processing facility (DPF)
3. the central processing control point (CPCP)
4. the operator console/operator (OPCON)

The actions described within the customer requirements document are:

1. BALLOON transmits coded message
2. DPF monitors active balloons
3. DPF notifies CPCP of confirmed balloon contact
4. CPCP sends request to DPF
5. OPCON send requests to DPF
6. DPF transmits information to CPCP.
7. DPF transmits information to DPF.
8. DPF purges information (as appropriate).
9. OPCON send data dump request to CPCP.

Derived or implied actions include:

10. DPF receives coded message from BALLOON.
11. DPF receives request message from CPCP.
12. DPF receives request message from CPCP.
13. CPCP receives request message from OPCON.
14. CPC performs data dump.

The derived actions above (#10 - #13) are the complement of an explicitly stated action (every transmission must have a receipt, etc.) Other actions (#14) may fill in omissions in the requirements statement or perform implied initialization or cleanup, etc. Unless careful guidelines and procedures are set to define and limit this effort, the identification of these derived actions may result in inconsistencies. Questions to be resolved include:

- Are Action #2 and Action #10 redundant?
- Should Action #14, indeed, be included?
- Should system initialization be considered?
- Is system maintenance (such as purging old records) required?
- What of error checking and handling?

For expediency, we will continue without resolving these questions.

The stated requirement is limited to developing a system for the DPF, so requirements placed on other objects are not relevant and are discarded. At this point, the resultant list of actions is:

1. DPF receives coded message from BALLOON.
2. DPF monitors active balloons
3. DPF notifies CPCP of confirmed balloon contact
4. DPF receives request message from CPCP.
5. DPF receives request message from OPCON.
6. DPF transmits information to CPCP.
7. DPF transmits information to DPF.
8. DPF purges information as appropriate.

There are potential interfaces among all objects within the system. The interfaces that involve the DPF are:

1. DPF --> CPCP
2. DPF --> OPCON
3. BALLOON --> DPF
4. CPCP --> DPF
5. OPCON --> DPF

The messages transmitted within the system are:

1. BALLOON --> DPF data transmittal
2. CPCP --> DPF report request
3. DPF --> CPCP requested report(s)
4. DPF --> CPCP balloon activation notification
5. DPF --> OPCON requested report(s)
6. OPCON --> CPCP dump request
7. OPCON --> DPF report request
8. OPCON --> DPF purge all request

We note that assumptions have been made. Although the CPCP makes two different requests of the DPF, these have been combined into a single message; however, the two requests from the OPCON to the DPF have not been combined. This is arbitrary and may be a source of both measurement and specification error.

Although design may well choose to combine multiple messages, all messages shall be separated for clarity and consistency. This may be considered a policy decision to establish a consistent procedure.

Discounting messages that don't impact the DPF and breaking any compound messages into separate messages, may result in the following:

1. BALLOON --> DPF data transmittal
2. CPCP --> DPF report for all balloons request
3. CPCP --> DPF report for a single balloon then purge request
4. DPF --> CPCP balloon activation notification
5. DPF --> CPCP requested report(s)
6. DPF --> OPCON requested report(s)
7. OPCON --> DPF report request
8. OPCON --> DPF purge all request

To this point the following simple (counting) metrics have been created:

- Number of Objects - 4
- Number of Actions - 8
- Number of Interfaces - 5
- Number of Messages - 8.

Differing viewpoints and interpretations will produce different requirements. Consider the five messages coming into the DPF:

1. BALLOON --> DPF data transmittal
2. CPCP --> DPF report for all balloons request
3. CPCP --> DPF report for a single balloon then purge request
7. OPCON --> DPF report request
8. OPCON --> DPF purge all request

Messages may be decomposed to more specific messages. Because weather data isn't gathered until the second transmission, message #1 might become two messages, #1A provides only balloon identification and position data and #1B includes weather information. Similarly, message #7 could be broken into two messages (analogous to messages #2 and #3), one requesting a report for a single balloon and one requesting a report for all balloons. The five original messages are thus decomposed into seven. Clearly, design decisions or interpretations are being made while gathering requirements.

Messages might be grouped. For example, messages #2 and #3 might be combined. If a message with the following structure is envisioned, then messages #2, #3, #7 and #8 can be combined into a single message:

REPORT REQUEST MESSAGE

FROM: (CPCP or OPCON)

WHICH BALLOON: (ID# or 0 for all)

PURGE: (YES or NO)

Therefore, from two to seven messages can be used to describe the same requirement. Message count, neglecting message content, may not be a sufficient metric.

The trivial example we are examining, has shown that the number of messages is a function of design and assumptions. Per the above discussion a metric that varies from two through seven, depending on how the engineer "sees" the system, is a problem. A set of guidelines must then be established. A working guideline might be that all messages are to be decomposed to the extent that no message may originate from or go to more than one object or group of similar objects. Thus, a message that goes to any one of a group (class) of similar entities (say a choice of like computers) would be considered a single message, but a message that goes to either of two dissimilar entities (say a computer and the operator), would be treated as two messages.

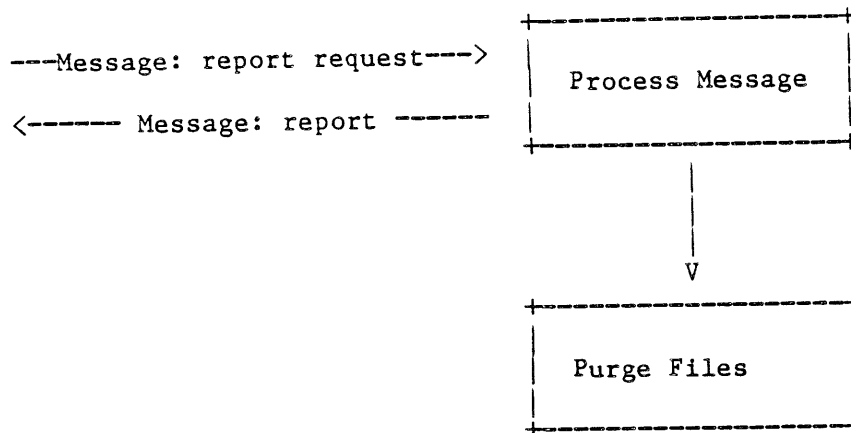
Data content is also problematic. A message that contains parameter data is not broken down on the basis of parameter values. A message whose record contents vary significantly, should be broken into separate messages. Note: "significantly" is not defined.

It is vital that any metric provide consistent, repeat-able measures. Thus, with our example, seven messages is the "proper" decomposition. Message Number 1 includes either (a) balloon identification and position data or (b) balloon identification and position data and weather information. There is a significant difference. Messages Number 2 and 3 also vary significantly; 2 simply includes a message type (report for all), 3 contains a message type (report/purge) and a field identifying which balloon. Message Number 3 is not broken any further because, although it selects a specific balloon to be purged, this selection is considered a parameter value (so the message is not decomposed into two messages). Clearly there is a judgement call involved. The resultant messages input to the DPF are: (Messages have been renumbered.)

1. BALLOON --> DPF initial transmittal (ID, Position)
2. BALLOON --> DPF data transmittal (ID, Position, data)
3. CPCP --> DPF report for all balloons request
4. CPCP --> DPF report for a single balloon then purge request
5. OPCON --> DPF report for all balloons request
6. OPCON --> DPF report for a single balloon then purge request
7. OPCON --> DPF purge all request

## Consistent Network Representation

System requirements form networks. Whether stated explicitly or not, there are networks of process flow and data flow. For example:



When analyzing a process (function) or record, decomposition is still an issue. Can the process or record be further decomposed or further grouped with other processes or records. During the requirements gathering, logical systems design should be unconstrained by physical systems limitations (i.e., the size of a function or record will not cause it to be decomposed.

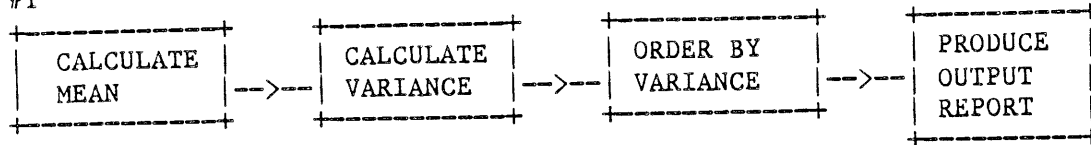
Consider the three following requirements statements:

1. There are 10 groups of 10 numbers each. The program shall determine the variances of each of the 10 groups and shall order the groups by minimum variance. The program shall determine the arithmetic mean of each group. An ordered report showing mean and variance shall be produced.
2. There are 10 groups of 10 numbers each. The program shall determine the mean and variance of each of the 10 groups and shall order the groups by minimum variance. An ordered report showing mean and variance shall be produced.
3. There are 10 groups of 10 numbers each. The program shall order the groups by minimum variance and produce an ordered report showing mean and variance.

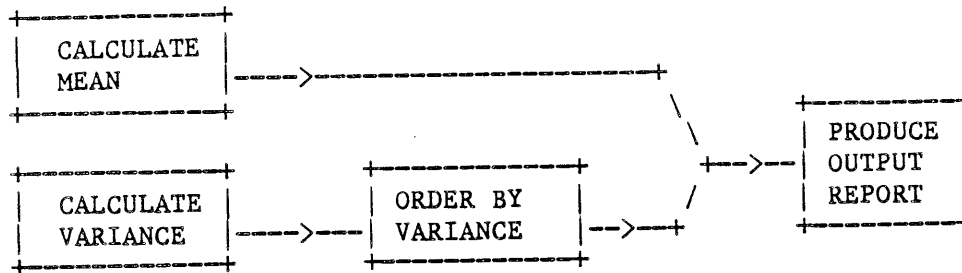
A requirements statement of each of these three requirements may produce different metrics. First of all, by using conjunctions the number of "Shalls" is different. Statement number three has an implied requirement (determine variance) which is stated explicitly in the other two statements.

Similarly, both the requirements statement and the resultant design may depict this requirement differently. Consider the three following networks:

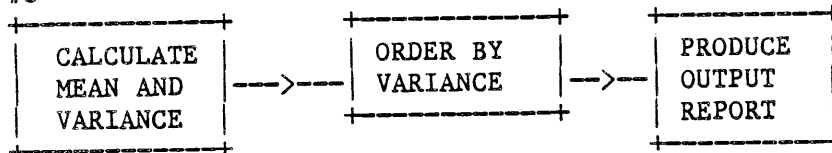
#1



#2

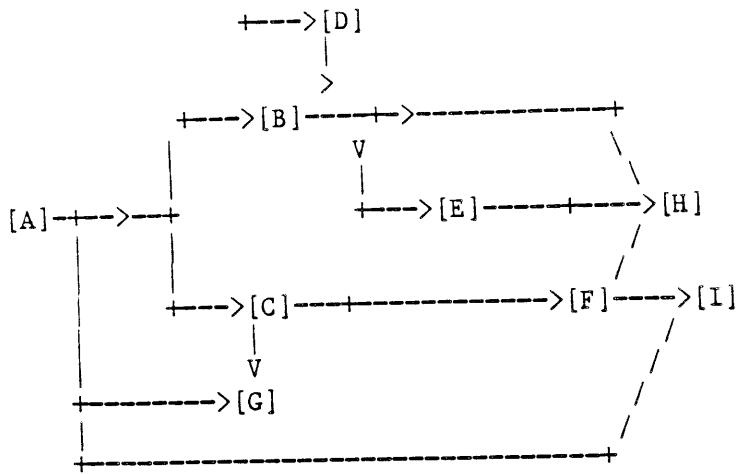


#3



In evaluating each of these representations we first note that the representation #1 shows that calculating variance must follow calculating mean. This comes from the classic mathematical definition of variance. Representation #2, perhaps realizing that there is a one pass algorithm for calculating variance or, not knowing anything about mean and variance (treating them as "black boxes) shows them as independent processes (or functions). Representation #3 may reflect knowledge of an existing, reusable, program that calculates both mean and variance. Representation #2 is "best" in that it reads the least external information into the system. Items are neither ordered nor grouped other than as required by the problem statement. If we were constrained by a computer with only 16 words of memory, we might wish to decompose this process - but that is a DESIGN, not a REQUIREMENTS STATEMENT prerogative. The constraint must be stated in the RSL, but the logical system definition must not be impacted. This introduces us to the area of measuring networks.

There are a number of most useful tools and metrics for analyzing networks and data structures. But here, again, consistency is difficult to attain. First we determine (as above) if the network clearly represents the requirement. We wish to state the requirement unambiguously and without overstating constraints. We must, therefore, determine if the network is well formed. The following example shows such a determination:



Read:  
 Process A must  
 precede Process C  
 (i.e. Data output  
 by [A] is required  
 input to [C])

First to allow mathematical manipulation we can express this network in matrix form using a precedence matrix.

$P_{ij} = 1$  if  $i \rightarrow j$   
 Precedence Matrix

	A	B	C	D	E	F	G	H	I
A		1	1				1		1
B				1	1			1	
C						1	1		
D									
E								1	
F								1	1
G									
H									
I									

Again, this is simply a method of depicting a network via a matrix.

Minimum Reachability determines the least number of steps between nodes.

$$R_{ij} = \min_k P_{ik} + P_{kj} \quad (\text{note: } P_{ii} = 0)$$

Minimum "Reachability" Matrix

	A	B	C	D	E	F	G	H	I
A		1	1	2	2	2	1	2	1
B				1	1			1	
C						1	1	2	2
D									
E								1	
F								1	1
G									
H									

The minimum "reachability" matrix takes this concept one step further, counting the minimum number of "steps" between any two network nodes.

(NOTE: Completeness checks, to determine if the network has any gaps, and consistency checks, to determine if the network has any loops follow as do aids such as clustering algorithms, commonality measures, etc. Any non-zero diagonal element indicates a loop; sources have a zero column total, sinks have a zero row total. Further discussion is, however, beyond the scope of this paper.)

In contrast, Maximum "Reachability" determines the longest number of steps between nodes.

$$R'_{ij} = \max_k P_{ik} - P_{kj}$$

Maximum Reachability Matrix

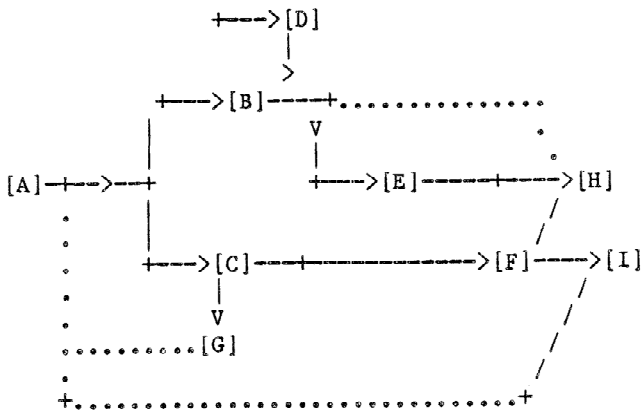
	A	B	C	D	E	F	G	H	I
A		1	1	2	2	2	2	3	3
B			.	1	1			2	
C						1	1	2	2
D								1	
E								1	1
F								1	
G									
H									
I									

If  $R'_{ij} < 1$  and  $P_{ij} = 1$ , then  $P_{ij}$  is an extraneous link and can be set to zero to reach "minimum data structure".

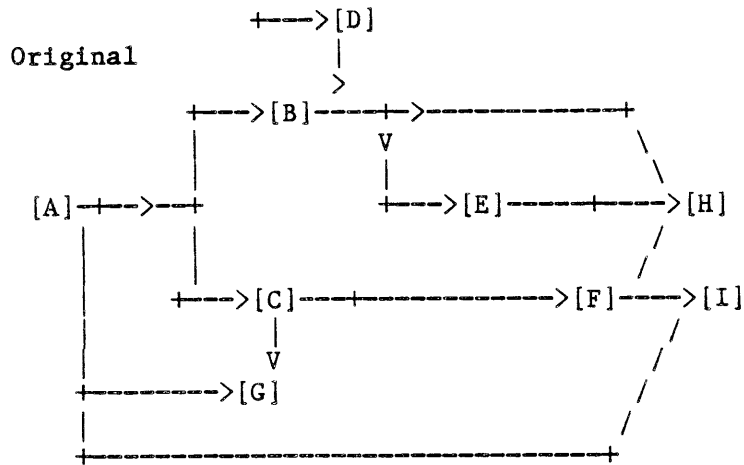
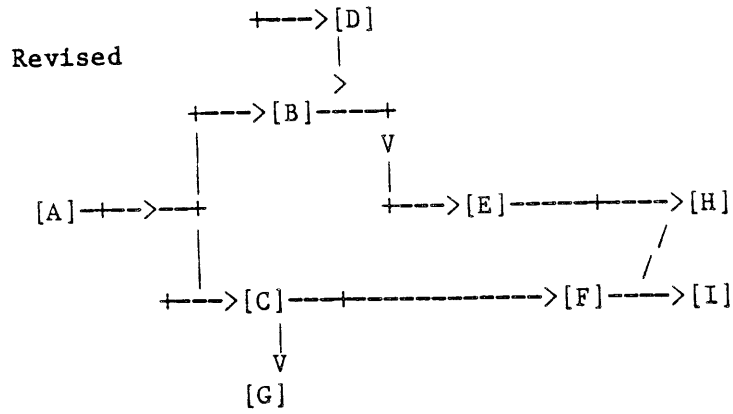
Revised Precedence Matrix

	A	B	C	D	E	F	G	H	I
A		1	1				.	.	.
B				1	1			.	.
C						1	1		
D									
E								1	
F								1	1
G									
H									
I									

The eliminated links are noted with '.' to contrast data structure with minimum data structure.



Compare the revised with the original:



Revised Minimum "Reachability" Matrix

	A	B	C	D	E	F	G	H	I
A		1	1	2	2	2	2	3	3
B				1	1			2	
C						1	1	2	2
D									
E								1	
F								1	1
G									
H									
I									

Revised Maximum "Reachability" Matrix (same as Minimum)

	A	B	C	D	E	F	G	H	I
A		1	1	2	2	2	2	3	3
B				1	1			2	
C						1	1	2	2
D									
E								1	
F								1	1
G									
H									
I									

Comparing the original network with the revised one:

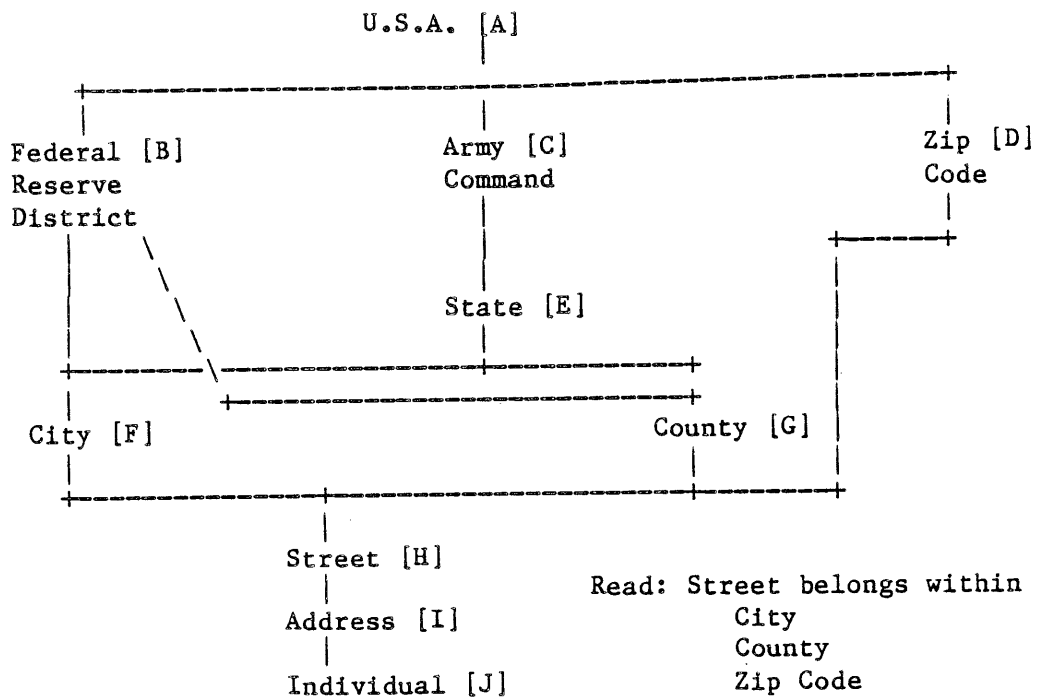
<u>Parameter</u>	<u>Original</u>	<u>Revised</u>
Nodes	9	9
Links	12	9
Min Complexity score*	24	29
Max Complexity score*	29	29

\* Score is simply the sum of the  $R_{ij}$ 's. Other measures are possible.

Thus, we have developed a consistent (if not optimal) way to depict a network. This same approach may be used for data structures by determining the minimum data structure.

### Data Base Design

Consider this data base schema:



### Precedence Matrix

	A	B	C	D	E	F	G	H	I	J
A	1	1	1							
B		1				1	1			
C			1							
D				1					1	
E					1	1				
F						1			1	
G							1			
H								1		
I									1	
J										1

## "Reachability Matrix"

	A	B	C	D	E	F	G	H	I	J
A		1	1	1	2	2	2	3	4	5
B						1	1	2	3	4
C					1	2	2	3	4	5
D								1	2	3
E						1	1	2	3	4
F								1	2	3
G								1	2	3
H									1	2
I										1
J										

In this case, further matrix analysis (not shown) found that this breakdown is, indeed, the minimal structure breakdown. In this manner, data base requirements and data base design can be supported by techniques for uniquely and correctly stating data relationships.

### Design

Design is a science and an art. Making decisions to realize efficiencies meet constraints. Design efforts include the decomposition, grouping and ordering of actions and of flows within a system as well as decomposition and grouping of data (data base design). Given the problem stated in RSL, the transition from requirements to design is constrained by the networking requirements (such as precedence and concurrency) which are developed and physical constraints including size, performance, etc.

We wish to measure design against the baseline formed by requirements and, in turn, we wish design to provide a suitable baseline for measuring the implementation activities that follow. Since design involves making choices, consistency will, of necessity, be lost.

Design involves tradeoffs as well, so the metrics measuring a design must reflect the tradeoffs made. Metrics reflect these tradeoffs, and metrics may also be useful design aids in making these tradeoffs. For example, a design decision to split a process into two concurrent sub-processes will reflect, in a higher process count, a lower average process size measure (possibly), a higher expected throughput rate and a higher (data) transport volume.

The system requirements (stated in an RSL) serve as the baseline for evaluating design. Design involves transforming the non-procedural logical systems specification into a procedural, physical systems specification. Since the logical systems specification is at a lowest level of detail, transforming from "non-procedural/logical" requirements statement to "procedural/physical" process design involves grouping processes together, ordering them (beyond that ordering demanded by network considerations) and assigning them to physical devices. Data base design involves grouping and structuring data (once again, as constrained by the requirements statement) and assigning data to physical devices.

The level of design detail may cloud metrics because the expanded detail is easily confused with a larger target system. For example, a high level design may state:

```
SORT THE CUSTOMER FILE
```

A more detailed (lower level) PDL description might be:

```
OPEN THE (INPUT) CUSTOMER_FILE
OPEN THE (OUTPUT) SORTED_CUSTOMER_FILE
SORT THE CUSTOMER FILE
WRITE THE (OUTPUT) SORTED_CUSTOMER_FILE
CLOSE THE CUSTOMER_FILE
CLOSE THE SORTED_CUSTOMER_FILE
```

Expressing design in a PDL will provide a useful data base for certain design measurement. The above design is expressed in PDL. The high level PDL shows only one function, the more detailed level shows seven functions. It does not, however, reflect a target system that is seven times as large. Consistent policies and procedures as well as a hierarchical PDL are required to minimize the impact of "level of detail" on design metrics.

### Conclusion

Requirements statement and design consist of gathering, ordering, grouping and decomposing functions and data items. Consistent and, therefore, useful metrics require that the levels of grouping and/or decomposition be comparable. As depicted by a formal requirements statement, mathematical tools and guidelines can be used to help develop consistent, comparable metrics of the target system.

It must be cautioned that the translation of requirements into design involves choices. To the extent that different choices are made, the same requirement document may translate to different formal requirements statements and then to different designs - thus, lacking consistency. Although design involves making choices within a constrained environment, useful and comparable (consistent) metrics of the target system can be developed.

The "metrician" is cautioned not to use an expedient, but improper, baseline: The "as requested" system, not the "as built" is the proper initial baseline.

Further work remaining includes:

- o detailed policies and procedures to accompany requirements statement languages and guide their use
- o well-coupled network tools to provide "in-line" checks of consistency, completeness and measures of connectivity and complexity
- o extensions to RSL's to provide better links for traceability, and project management and control
- o an easy to use, hierarchical PDL to communicate multiple levels of design detail

APPENDIX A: An RSL Reading list

Alford, Mack. SREM at the Age of Eight; The Distributed Computing Design System. TRW Huntsville Laboratory. IEEE COMPUTER, April 1985.

Bell, Thomas E.; Bixler, David C.; Dyer, Margaret E., An Extendable Approach to Computer-Aided Software Requirements Engineering. IEEE Transactions on Software Engineering, Volume SE-3, Number 1, January 1977.

Lynch, H. J., ADS: A Technique for Systems Documentation, Database Volume 1, Number 1, Spring 1969.

National Cash Register Company, Accurately Defined Systems, 1967.

Nunamaker, J. F., A Methodology for the Design and Optimization of Information Processing Systems, Krannert School of Business, Paper Number 301, Purdue University, March 1971.

Nunamaker, J. F., Ho, Thomas; Konsysnski, Benn; Singer, Carl A.; Analysis and Design of Information Systems. Journal of the ACM, December 1976.

Singer, Carl A., A Methodology for the Determination and Communication of Requirements for an Information Processing System, PhD Dissertation, The Krannert School of Business, Purdue University 1975.

Steiger, W. H., The Communications Problem in Systems Building, ISDOS (Information Systems Design and Optimization System) Working Paper Number 2, Case Western Reserve University, September 1967.

Teichroew, Daniel; Sibley, Edgar H.; Metrick, Lee B.; Sayani, Hasan; PSL - A Problem Statement Language for Information Processing Systems Design, ISDOS Project, University of Michigan, 1969.

Teichroew, Daniel; Hershey, Ernest A., III; Bastarache, Michel J.; An Introduction to PSL/PSA, ISDOS W.P. No. 86, March 1974., University of Michigan.

Teichroew, Daniel, A Survey of Languages for Stating Requirements for Computer-Based Information Systems, FJCC, 1972.