# Improved Software Reliability Through Requirements Verification

Kurt F. Fischer
Computer Sciences Corporation, Falls Church
Michael G. Walker
Computer Sciences Corporation, Falls Church

*Key Words*–Software reliability, Requirement verification

*Reader Aids*–
Purpose: Widen state of the art
Special math needed: Graph theory is helpful
Results useful to: System and software engineers

*Abstract*–Requirement errors discovered early in the development process are several orders of magnitude less expensive to fix than if these same errors are discovered late in the development process. Therefore, it is expedient to investigate methods for discovering requirement errors early in the development process.

This article decribes a technique to improve software reliability through verifying requirements early in the software development process. The technique involves the generation of a System Verification Diagram (SVD) for each set of functional requirements. The prime purpose of developing SVDs is to serve as a tool for verifying the functional consistency and completeness of a requirement specification. In so doing, it also becomes an excellent communication device for reviewing the requirements to assure that customer needs are not inaccurately portrayed.

## I. INTRODUCTION

### A. Background

Software reliability has received much attention in the engineering and computer science literature beginning with the IEEE Symposium on Computer Software Reliability in 1973 [1]. Since then, software reliability has become a major topic at conferences sponsored by the IEEE, the Association for Computing Machinery (ACM), the American Institute of Aeronautics and Astronautics (AIAA), the Microwave Research Institute (MRI), and many other academic, industrial, and governmental organizations. This attention can be attributed to both a clearer understanding of the problem (i.e., the continued development of unreliable software), and a better understanding of potential solutions (e.g., rigorous specification methods, structured approaches to software development, formal verification methods).

With the ever increasing proportion of total system budgets going to software [2], a continuing investigation of methods for increasing software reliability will be of high importance. If one calculates software reliability as a function of the number of errors in a software system (as most people do), then reliability can be increased by reducing the number of software errors. The problem thus becomes one of locating and repairing the errors in the most cost-effective manner.

Most software systems have been built following a series of life-cycle phases which basically consist of:

- Requirements definition
- Design
- Programming
- Integration and test

Surprisingly, during the development of many of these systems few attempts are made to verify the products of each intermittent phase (i.e., errors introduced during the requirements, design, and programming phases are not discovered until later in the test phase). Frequently, software specialists do not recognize that many problems discovered in later life-cycle phases are requirements oriented. However, Bell & Thayer [3] have published data showing that requirement errors can be a significant problem to software development. Their study on one large real-time system turned up 972 uniquely identifiable problems reported against 8248 individual requirement and support paragraphs. Boehm's survey [4] has shown that the cost of fixing errors increases expoentially as one progresses further into the software life-cycle. Therefore, the cost of developing software could be greatly reduced by finding errors during the life-cycle phase in which they are introduced.

This paper discusses one method of detecting requirement errors during the requirement definition phase. The technique, called System Verification Diagrams (SVDs), has been used successfully to verify both system-level and software-level requirements. Though Computer Sciences Corporation has used this technique on many projects, other techniques for stating and verifying requirements have also been published [5 - 7]. After some introductory remarks regarding requirements verification in this section, the SVD technique will be explained in Section II, and some experience regarding its use will be presented in Section III.

### B. Requirements Verification

The process of reviewing software requirements to find errors is called requirements verification. The traditional method for this procedure is a formal system requirements review. The procedure involves 1) publication of the system requirements, and 2) an independent review by design engineers, test engineers, and customer representatives to ascertain that the requirements adequately and accurately meet the mission or product objectives. While this review is better than no review, it has two major difficulties:

1. The original requirements were defined and written using sound engineering judgment. Therefore, why should

0018-9529/79/0800-233 $00.75 

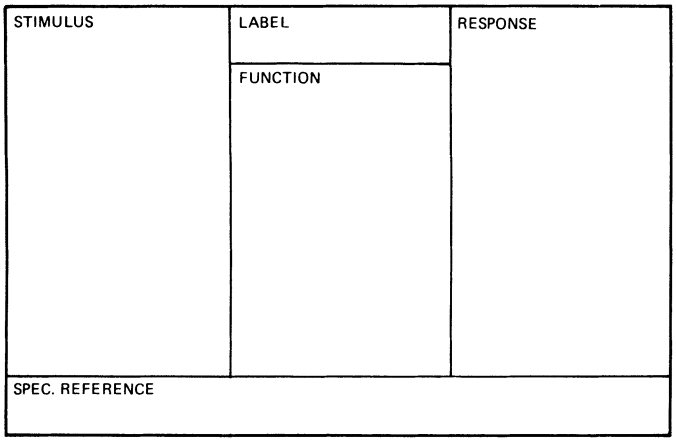


Fig. 1. Decomposition Element Form

a review, also using sound engineering judgment, be very effective?

2. Customer representatives are not usually as skilled at defining the problem in as succinct terms as the contractor's system engineers (otherwise, why would the customer hire the contractor). Often, the customer has difficulty in comprehending the system specification, and not until he sees the system unfolding (during the design, programming, and test phases) does he realize that what is being developed does not suit his needs.

To counter these problems, CSC developed in 1973 a technique for requirements verification called System Verification Diagrams (SVDs). This technique aids the system engineer in requirements verification by displaying the functional requirements as a directed graph. Quantitative analytic procedures can then be applied to the directed graph to allow the system engineer to verify the clarity, completeness, consistency, and traceability of the functional requirements. In addition, the directed graph presents the functional requirements in a form that is easier to comprehend than several hundred (or thousand) pages of technical detail, thus providing customer representatives useful feedback of the system engineer's understanding of the customer's problem.

## II. SYSTEM VERIFICATION DIAGRAMS

A system can be represented by a model consisting of 1) the stimuli or inputs to the system, 2) the system functions or processes that operate on these stimuli, and 3) the responses or outputs resulting from the functions. A given set of stimuli produces a certain response, and the processing can be treated as a black box. The SVD technique is based on decomposing functional requirements into units called decomposition elements (DEs). Each DE identifies the stimulus and response of each function in the system. The DEs are then graphically displayed showing all stimulus/response flows through the system. This graphical mapping is called an SVD, which is defined as a directed graph of decomposition elements representing the overall system functional description.

The prime purpose of an SVD is to serve as a tool for verifying the clarity, consistency, completeness, and traceability of functional system or software requirements. Thus, an SVD can be used during the system requirement definition phase (prior to system design) or during the software requirement definition phase (prior to software design). There are three steps in the SVD process: A) decomposition, B) graph preparation, and C) graph analysis. Each of these steps is covered below.

### *A. Decomposition*

Preparation of the SVD begins with a review of the applicable system/subsystem requirement specification and the translation into its decomposition elements. It is a step-by-step process involving conversion of the specification text into unique functions, each with its own identifiable stimuli (inputs) and responses (outputs). As the requirements are reviewed, key words and phrases in each process (or functional) description are highlighted. As the functions are identified so too are the stimuli and responses associated with that function.

Each function, along with its stimuli and responses, is documented on a decomposition element form (Fig. 1). The definitions of the contents of the decomposition form are provided below.

STIMULUS – External system/subsystem input from any source, whether manual or automated. This includes operator request/command, user terminal, system message, external file. An input could also be defined as a system condition that initiates the process.

FUNCTION – A functional definition of a unique system/subsystem process with well defined input and output. This block should identify what the process is. It should not describe those process components that modify or constrain the process.

RESPONSE – All products of the process, including any unique system conditions, that are the direct result of the defined function. This includes messages, reports, listings, system status.

LABEL – A unique numerical identifier assigned to each decomposition element in the SVD. There is no convention for assigning the identifiers except that they should follow a numerical sequence within each segment or subdivision of the system/subsystem SVD.

REFERENCE – Specification/manual paragraph number(s) where the functional requirement is defined or identified.

In order to illustrate how the process descriptions are identified, an excerpt from a subsystem specification document is shown in Fig. 2. The text shows a portion of the description of general requirements for display processing at the subsystem level within an automated air traffic control system. Key process descriptions, words, and phrases can be identified after reading the material carefully. This figure illustrates the highlighting (via underline) of key words and phrases once they are identified. The part of the text which describes filtering and

3.7.3.1.3.1.1.6 Display of Radar Data

Primary and secondary radar data shall be made available for display after passing the correlation process and provided that correlation with an existing track was not possible. These data shall be subjected to both a filtering and routing process such that displayed radar data shall be limited to a single radar source per geographic area and only that radar data which must be displayed on a particular CRT is routed to that CRT from the display generator in the computer.

Limiting of radar data for display purposes shall be accomplished by assigning a preferred and supplementary radar to each geographic area. Only those radar plots within a geographic area that originated from the preferred radar shall be eligible for display. The size and quantity of the geographic areas shall be determined by the contractor and shall be alterable through adaptation data. The supplementary radar shall be used only is the preferred radar is inoperative.

The following categories of track data shall be displayed on the CRT.

1. Tracks and uncorrelated plots, including histories
2. All plots (backup mode when tracking is inoperative)
3. Test and simulation information:
   a. All correlated tracks and plots, including histories
   b. All plots and tracks

Functional Units shall be provided with switch actions which permit the controllers to specify the categories of radar data which they wish displayed on their CRT. Controllers shall have the capability to specify for display radar data in categories 1 and 2 above.

3.7.3.1.3.1.1.6.1 Horizontal Display Selection

The optimum display of sectors on the CRT is effected by controller input of the display center point and display scale. Capability shall be provided for the controller to make these inputs via switch action request. These inputs shall be stored in the processor so that, if the center point and scale are changed during the course of operation, the basic setting can be returned by switch action request. It shall be possible to displace the center point by use of the rollball. All tracks/plots which lie within the horizontal display area of the screen shall be displayed.

3.7.3.1.3.1.1.6.2 Vertical Display Selection

Four altitude borders apply as follows:

1. Upper buffer boundary
2. Sector boundary (top)
3. Sector boundary (lower)
4. Lower buffer boundary

All plots/tracks shall be displayed which lie between the upper and lower altitude buffer boundaries.

3.7.3.1.3.1.1.6.3 Display Labeling

Radar data shall be displayed utilizing different symbols. In response to an air traffic controller's switch action request, the computer program shall effect the generation of plot/track histories on the CRT of the requesting Functional Unit. The number of history points shall be specifiable for each CRT via controller switch action and shall range from 1 to 6 (excluding the head symbol). The head symbol shall be displayed with at least 2.5 times the brightness of the history symbols. An automatic aging logic shall be incorporated. This aging logic shall automatically replace the current head symbol by the newest plot/track position data, and increase the age of all history symbols, dropping the history point of greatest age. The history points for plots shall be displayed in a chronological sequence, with the current radar data being displayed last. The display cycle for radar track output data shall be updated every 5 to 10 seconds. Plot data shall be displayed as received. For each uncorrelated plot having a valid code, the code shall be displayed adjacent to the uncorrelated plot display symbol.

For each plot containing an emergency indication, a unique symbol shall be displayed.

Fig. 2. Example of Highlighted Requirements Specification

routing is not highlighted as filtering and routing describe 'how' the process is accomplished, not 'what' is being performed.

The translation of specification text to decomposition elements begins after the system/subsystem specification document has been reviewed completely and all process data are highlighted. Using the highlighted specification text, key words and phrases are extracted to identify the stimuli, processing functions, and responses that are placed into the decomposition element forms (Fig. 1). For example, the highlighted material shown in Fig. 2 provides data for the DEs that are shown in Fig. 4. (Fig. 4 also shows the DE connectivity relationships which will be subsequently discussed.) When extracting key words and phrases from the specification text, the phraseology of the specification should be retained in the DE blocks. This creates an SVD that more easily traces to the specification document and facilitates relating the DEs to the specification after the graph is completed. The stimulus wording should be generalized or itemized to include all possible stimuli to the function. The function wording should begin with a verb that denotes action. The response wording

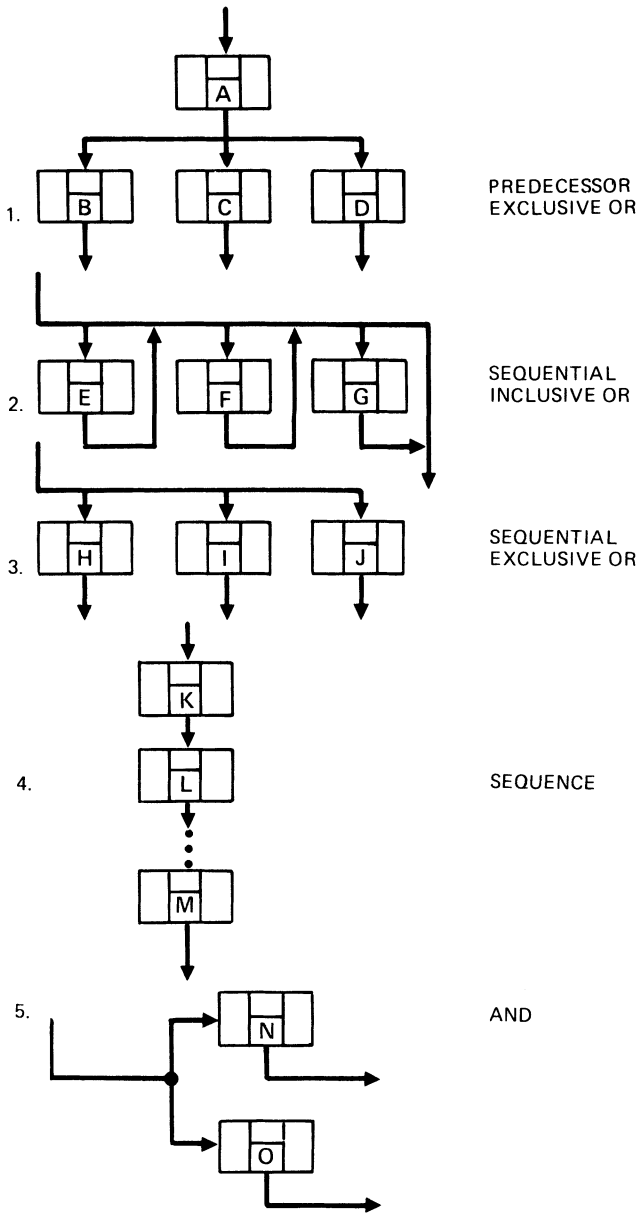


Fig. 3. Decomposition Element Connectivity Relationships

should include all products and/or links to other functions. In that respect, when the DEs are placed within the directed graph structure, stimulus and response wording may require modification so that continuity between and among decomposition elements is maintained.

### B. Graph Preparation

The graph preparation stage begins when all system/subsystem functional requirements are translated into decomposition elements. This process has recently been aided by automation, but it will be explained as if done manually to illustrate the process more clearly.

The individual DEs should be sorted so that subsets common to a particular function or segment are together. Using a large, flat area (cork board or table), each subset is laid out to form a trial functional structure. Each DE can be logically related to an adjacent DE or to a set of DEs in one of five ways. These five permissible relationships are illustrated in Fig. 3 and described below.

1. *Exclusive OR:* Decomposition element *A* is the predecessor of elements *B, C, D.* Elements *B, C, D* are connected by a common bus and are 'exclusive OR' successors of *A.* This means that, following the processing associated with the function in element *A,* either *B* or *C* or *D* (only one) will occur, depending upon the inputs to the elements that follow element *A.*
2. *Sequential Inclusive OR:* Elements *E, F, G* are 'sequential inclusive OR' successors, meaning that the input stimulus to element *E* is checked first to see whether or not it exists (i.e., whether or not it is generated as an output of some other *DE* or is a global stimulus. If the input stimulus to element *E* does exist, then the functional processing associated with element *E* is performed. Regardless of whether element *E* was processed, the input stimulus to element *F* is checked, followed by the input stimulus to element *G.* Any combination of elements *E, F, G* may occur.
3. *Sequential Exclusive OR:* Elements *H, I, J* represent 'sequential exclusive OR' successors, meaning that the input stimulus to element *H* is checked before *I* and *I* is checked before *J,* but that the first match is the only one of the three to occur.
4. *Sequence:* Elements *K, L, M* represent 'sequence,' meaning that *K* must occur before *L,* and that L must occur before *M.*
5. *AND:* Elements *N, O* represent the start of parallel processes, both of which will occur.

In the process of developing an SVD, it may be necessary to change *DE* wording and/or delete or combine other *DE*s. The object is to show logical connectivity among the functions contained in the requirement specification. Once a trial logic structure is achieved that appears to reflect the functional flow, a draft SVD can be created. The SVD for the text presented in Fig. 2 is illustrated in Fig. 4.

### C. Graph Analysis

The *DE*s, following their preparation, can be used in both a manual and an automated analysis of the requirement specification. Manual analysis helps the system engineer assure both completeness and consistency; their definitions are:

*Completeness*—A requirement is complete when a *DE* has been created for every possibility inherent in the specification text. For example, if the text specifies three variables, each of which may have an on-off state, then $2^3 = 8$ *DE*s must be produced to account for all possible alternatives. If the text specifies less than 8 possible outcomes, then it is incomplete.

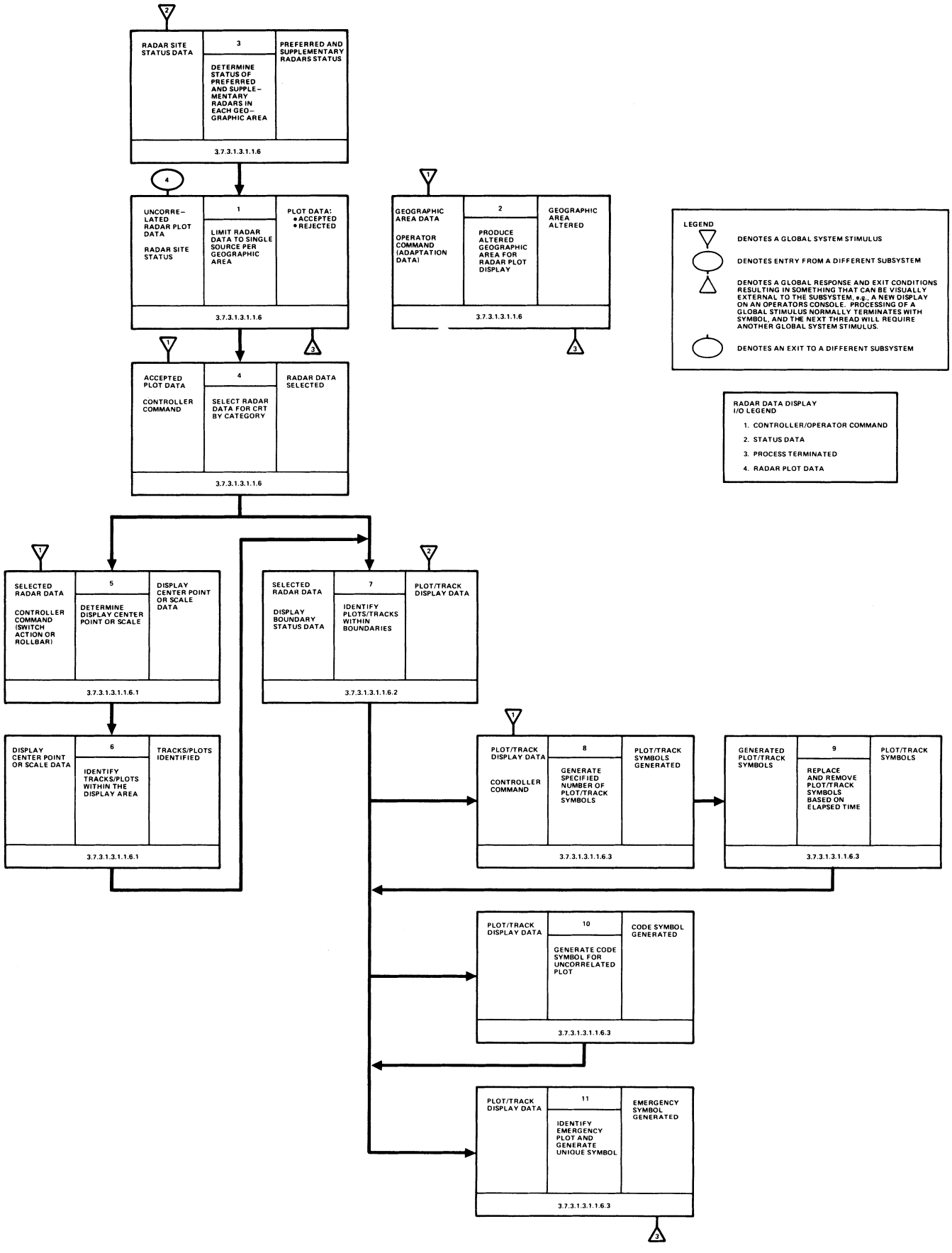


Fig. 4. Sample System Verification Diagram

| FROM \ TO | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig. 5. Connectivity Matrix

| FROM \ TO | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig. 6. Reachability Matrix

*Consistency*—A requirement is consistent if, for each time it specifies a stimulus within a condition, the same response is elicited. For example, a requirement might state that, when button $A$ is pressed on a console, light $X$ and only light $X$ shall be illuminated. This is a stimulus eliciting a response. If at any other location the requirement specifies that button $A$ must illuminate light $Y$, then the requirement specification is inconsistent. The inconsistency might not be immediately obvious, but might only be determined by analyzing the reachability matrix (defined later).

Automation allows the accurate analysis of large SVDs very quickly and inexpensively. Much of the following explanation is taken from [8]; since this area is stell being investigated, this explanation merely introduces this kind of analysis. The previous example, consisting of 11 *DE*s from the SVD shown in Fig. 4, illustrates the analytic procedure.

Automated analysis begins with the generation of connectivity and reachability matrices. The functional connectivity matrix represents the logical connections among the decomposition elements. This is an $N \times N$ matrix (where $N$ equals the number of *DE*s in the SVD) where any $(i, j)$ element is 1 if $DE_i$ transfers directly to $DE_j$; 0 otherwise. The connectivity matrix of SVD functions (*DE*s) from Fig. 4 is shown in Fig. 5. This matrix can be manually verified (for example, $DE_3$ transfers to $DE_1$); therefore, the (3, 1) position contains a 1, etc.

The following observations can be made from the functional connectivity matrix:

- Any column of all zeros identifies a *DE* that is not transferred to from any other *DE*, and therefore must be an entry *DE*. Both $DE_2$ and $DE_3$ are entry *DE*s.
- Any row of all zeros identifies a *DE* that does not transfer to any other *DE*, and therefore must be terminal *DE*. Both $DE_2$ and $DE_{11}$ are terminal *DE*s.
- If there is a one along the diagonal, it indicates that a particular *DE* loops back to itself. Though possible, this is improbable at the requirement level and usually indicates an error.
- Any *DE* with an associated zero row and zero column indicates a stand-alone system function. Though possible (as is $DE_2$), they are rare and should be double-checked for authenticity.

Directed graph methods have been used to analyze computer programs for some time [9, 10] but the ability to analyze requirements with these techniques is novel. For example, early in the development cycle it would be very useful, for planning purposes, to have an estimate of the required number of system tests. Directed graph analysis of the requirements can help by providing both an upper and a lower bound on the required number of system tests. Sloane's algorithm [11] can be used to estimate the upper bound by generating all the paths through a graph. Lipow's extension to Dilworth's theorem [12] can be used to generate the lower bound of system tests by identifying the minimum number of tests necessary for at least a check of each transfer between *DE*s at least once.

These techniques can be used to help estimate the system test effort as early as the requirement definition phase. One can estimate the required number of system tests by taking the mean of the upper and lower bounds, or one can adjust that estimate depending upon the desired thoroughness of the test effort.

Automated analysis techniques are also useful during the maintenance phase to assess potential modification impacts by using the reachability matrix. Since changes to one function often impact the performance of other functions, all *DE*s reachable from the modified *DE* should be retested. This can be determined from the reacability matrix. The reachability matrix has size $N \times N$ where any $(i, j)$ element is 1 if $DE_j$ could ever be reached (directly or indirectly) from $DE_i$; 0 otherwise. The reachability matrix of the SVD functions (*DE*s) from Fig. 4 is shown in Fig. 6. For example, if a change is made to the requirements associated with $DE_2$, only $DE_2$ would have to be retested. On the other hand, if a change is made to the requirements associated with $DE_7$, then $DE_7$, $DE_8$, $DE_9$, $DE_{10}$, $DE_{11}$ would have to be retested. This eliminates much of the guesswork associated with maintenance retesting.

In summary, the graph theory for requirement analysis is most useful. Much of the graph theoretical research on computer code may apply to requirements also, and this possibility is being investigated.

## III. CONCLUSION

System Verification Diagrams have proved very effective for detecting requirement errors early in the life cycle. Data have been collected from five projects regarding the use of SVDs. These data were given to the authors as a 'best estimate' by the senior project engineer based on his direct experience. Characteristics of each project are shown in Table 1. Project E has yet to be coded, but is expected to take 500-700 kilolines of source code. Fig. 7 shows the estimated fraction of total requirement errors detected as a direct result of using SVDs to verify the requirements after they had been (what was thought to be) adequately defined and documented. The table shows that, in most cases, projects can detect 75 to 85 percent of the requirement errors while still in the requirement definition phase by applying a requirements verification technique. These data were given to the authors by the senior project engineer as a 'best estimate' based on his direct experience. Project B, which had the lowest estimated percentage of requirement errors found during the requirement verification process, was a project whose customer would not review the SVDs. The price that he paid was that a significant number of errors were not caught until later in the life cycle, and fixed at a presumably higher cost.

Experience shows that SVDs can detect three types of errors:

- Inconsistencies.
- Incompleteness.
- Misinterpretations.

Figure 8 shows the distribution of defect categories for those errors detected during the requirement verification process.

Requirement verification using SVDs has been beneficial to Computer Sciences Corporation. For example, on one recent medium-sized logistic project, the SVDs were delivered as part of the proposal and refined during the requirement definition phase. During the system requirements review, the customer felt he understood the system requirements much better with the aid of SVDs and requested that they be baselined with the system specification. As a result of the customer's and the vendor's clear understanding of the requirements, there were no requirement disagreements during either software development or acceptance.

Top management at Computer Sciences Corporation has felt the need for requirement verification so strongly that it has included it as part of a corporate software development policy. Current and potential customers are actively encouraged to add a requirement verification task to requirement definition efforts in order to increase software reliability, improve customer understandability, and reduce life-cycle costs.

## ACKNOWLEDGEMENT

A special thanks goes to D.H. Cook for giving us added in insight into SVDs, J.D. Donahoo who supplied the SVD example, P.C. Belford under whose direction some of the directed graph analysis was developed, and R.J. Carey who provided the SVD error data.

Table 1
Characteristics of Projects From Which SVD Data Was Collected

| Project I.D. | Project Description | Computer Language | Estimated *K* Lines of Code |
|---|---|---|---|
| A | Shipboard tactical command center | CMS-2 | 150 |
| B | Ground-based tactical command center | JOVIAL | 350-500 |
| C | Shipboard combat weapons system | CMS-2 | 1 000 |
| D | Shipboard communication system | Assembly | 350 |
| E | Air traffic control | Undetermined | 500-700 |

| Project | Percent |
|---|---|
| A | 85 |
| B | 55 |
| C | 75 |
| D | 85 |

Fig. 7. Estimated Percentage of Requirement Errors Detected Using SVDs.

| Project | Inconsistency (%) | Incompleteness (%) | Misinterpretation (%) |
|---|---|---|---|
| A | 20 | 30 | 50 |
| B | 20 | 40 | 40 |
| C | 40 | 30 | 30 |
| D | 25 | 35 | 40 |
| E | 35 | 30 | 35 |

Fig. 8. Requirement Error Fractions by Error Category

## REFERENCES

[1] *Record, IEEE Symp. Computer Software Reliability,* IEEE Cat. No. 73-CH0741-9CSR, 1973 April 30 - May 2.

[2] B.W. Boehm, "Software and its impact: A quantitative assessment", *Datamation,* 1973 May, pp 48-59.

[3] T.E. Bell, T.A. Thayer, "Software requirements: Are they really a problem?" *Proc. Second International Conf. Software Engineering,* IEEE Cat. No. 76CH1125-4C, 1976 October.

[4] B.W. Boehm, "Software engineering", *IEEE Trans. Computers,* vol C-25, 1976 December, pp 1226-1241.

[5] D. Teichroew, E.A. Hershey III, "PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems", *IEEE Trans. Software Engineering,* vol SE-3, 1977 January, pp 41-48.

[6] M.W. Alford, "A requirements engineering methodology for real-time processing requirements", *IEEE Trans. Software Engineering,* vol SE-3, 1977 January, pp 60-69.

[7] D.T. Ross, K.E. Schoman Jr., "Structured analysis for requirements definition", *IEEE Trans. Software Engineering,* vol SE-3, 1977 January, pp 6-15.

[8] *Systems Decomposition Technology,* Computer Sciences Corporation, CDRL A007 under Contract DASG 60-75-C-0125 to U.S. Army, BMDATC, 1976 April.

[9] R.B. Marimont, "Applications of graphs and boolean matrices to computer programming", *SIAM Reviews,* vol 2, 1960 October, pp 259-268.

[10] R.M. Karp, "A note on the application of graph theory to digital computer programming", *Information and Control,* vol 3, 1960, pp 179-190.
[11] N.J.A. Sloane, "On finding the paths through a network", *Bell System Technical Journal,* vol 51, 1972 Feb. pp 371-390.
[12] M. Lipow, "Some directed graph methods for analyzing computer programs", *Computer Science and Statistics: Eighth Ann. Symp. Interface,* UCLA, 1975 February.

## AUTHORS

Kurt F. Fischer; Computer Sciences Corp.; 6565 Arlington Blvd.; Falls Church, VA 20046 USA.

**Kurt F. Fischer** is a Senior Computer Scientist at Computer Sciences Corporation (CSC). He is Manager of the Program Management and Product Assurance Department with responsibility for independent audits of major software projects, and for the development of CSC's software development methodology: the Digital Systems Development Methodology. Previously, he was with TRW as Manager of software quality assurance tools and as a Software Quality Assurance Engineer. Mr. Fischer has a BS in Quantitative Methods from the University of Illinois, an MS in Management Science from Colorado State University, and is a Doctoral Candidate at UCLA's Graduate School of Management. He has previously published with IEEE, ACM, and ASQC.

Michael G. Walker; Computer Sciences Corp.; 6565 Arlington Blvd.; Fals Church, VA 20046 USA.

**Michael G. Walker** is a Senior Computer Scientist at Computer Sciences Corporation. He is responsible for software quality assurance on a project to develop a large computer information system. Previously, he was employed by the U.S. House of Representatives where he developed the Summary of Debate System and a budget tracking system for six Congressional Oversight Committees. He received a BA in Psychology from Bowdoin, an MA in Management Science and a PhD in Public Administration from the American University in Washington, DC. He is author of several articles and of the book from Elsevier North Holland: *Managing Software Reliability: The Paradigmatic Approach.*

Manuscript SI79-09 received 1978 December 10; revised 1979 January 31.

☆☆☆

# Book Reviews

**Ralph A. Evans, Product Assurance Consultant**

### Microcircuit Screening Effectiveness, TRS-1

Henry C. Rickers, 1978, $30 (nonUSA $36), 104 pp. Reliability Analysis Center; Rome Air Development Center (RBRAC); Griffiss AFB, NY 13441 USA.

Table of Contents



This is generally a good report, although all of the explanations must be read very carefully in order to understand exactly what the charts mean, and don't mean. The biggest difficulty with using "fraction failed" as any measure of screening effectiveness is that it does not consider two things:

1) The intent of the user might not have been to select those units having certain survival properties, but rather something else. For example, in the extreme case, some people have taken only the central 10% or so of the population being screened; because they wanted to be sure they didn't get any mavericks.

2) The false rejects might be important -- those which would have survived the actual use, but did not survive the screen.

At least some of the charts consider failure-modes (rather than total failed units) which were removed.

None of these comments are necessarily adverse. They are intended to make the user of the Handbook very aware of what was actually done and what the reports actually say. Burn-in screens is a very tricky subject, and it is handled reasonably well in this report. There is much good engineering information in addition to the data. This is a report to be studied carefully before any of the numbers are used. If that is done, the report can be very valuable. If not, you can mislead yourself a long way.

### Search and Retrieval Index to IRPS Proceedings - 1968 to 1978, TRS-2

Roy C. Walker, 1979, $20 (nonUSA $24), 384 pp. Reliability Analysis Center; Rome Air Development Center (RBRAC); Griffiss AFB, NY 13441 USA.

Table of Contents



The IRPS is the International Reliability Physics Symposium and is the major symposium in the field. This report is a large indexing job and RAC put a great deal of useful work into it. The papers are indexed in many ways -- all with an eye to getting the most from the Proceedings. You don't have to read the explanations of the indexing techniques, but you will have well spent the few minutes if you do.

These Proceedings are not for everyone; don't confuse them with the Proceedings of the Annual Reliability and Maintainability Symposium. But if you are in the field of failure modes and mechanisms of semiconductors, then you ought to be familiar with these Proceedings, and probably ought to have access to this report. It tells you how to get microfiche copies of the older Proceedings, and where to order copies of the more recent ones. If you want to get these Proceedings regularly, join one of the groups that sponsor this Transactions/Journal.