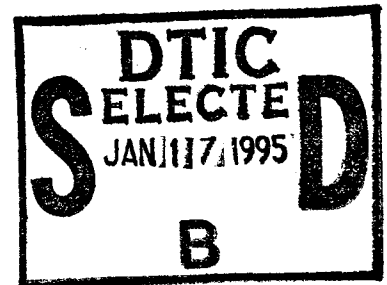RL-TR-94-201, Vol II (of two)
Final Technical Report
November 1994

VERIFICATION AND VALIDATION GUIDELINES FOR HIGH INTEGRITY SYSTEMS

SoHaR Incorporated

Herbert Hecht, Myron Hecht, George Dinsmore, Sara Hecht,
and Dong Tang

DTIC
S ELECTE D
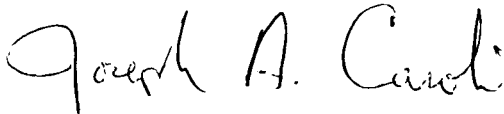JAN 17 1995
B

DTIC QUALITY INSPECTED 8

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

19950113 021

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-201, Volume II (of two), has been reviewed and is approved for publication.

APPROVED: *Joseph A. Caroli*

JOSEPH A. CAROLI
Project Engineer

FOR THE COMMANDER: *John J. Bart*

JOHN J. BART
Chief Scientist, Reliability Sciences
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (ERSR) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE November 1994	3. REPORT TYPE AND DATES COVERED Final Jul 92 - Jul 94

4. TITLE AND SUBTITLE VERIFICATION AND VALIDATION GUIDELINES FOR HIGH INTEGRITY SYSTEMS	5. FUNDING NUMBERS C - F30602-89-D-0099, Task 8

6. AUTHOR(S) Herbert Hecht, Myron Hecht, George Dinsmore, Sara Hecht, and Dong Tang	PE - 62702F PR - R429 TA - QA WU - 10

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SoHaR Incorporated 8421 Wilshire Blvd, Suite 201 Beverly Hills CA 90211-3204	8. PERFORMING ORGANIZATION REPORT NUMBER N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (ERSR) 525 Brooks Rd Griffiss AFB NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-94-201, Vol II (of two)

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Joseph A. Caroli/ERSR/ (315) 330-4205

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Software verification and validation (V&V) methodologies were investigated for high integrity systems. The effort was jointly sponsored by the Nuclear Regulatory Commission and the Electric Power Research Institute as a precursor to official nuclear regulatory guidance. The technology is dual-use; both the nuclear and defense communities will benefit from it. A risk-based classification scheme was developed for high integrity systems. Risk is made a function of probability of failure and the severity of consequences associated with failure. The intent is to tailor the level of V&V according to the classification. An error classification scheme was developed for use with a database of failure modes and failure rates of existing systems. The intent is to enable a more objective evaluation of software V&V methodologies. The application of quality metrics to the high integrity software development process was explored. High integrity software verification guidance was developed for several methodologies, including reviews and audits, independent equivalent activities, backward reconstruction and algebraic methods. Verification over the entire life cycle is addressed. Two new tools, the Enhanced Condition Table Tool and the Code Analyzer Tool Set, are explored as methods of verification completely independent of the software developer. A combination of functional, structural, and statistical testing is (see reverse)

14. SUBJECT TERMS Software quality, Software reliability, Software fault tolerance, Nuclear safety systems, Verification, Validation (see reverse)	15. NUMBER OF PAGES 52
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

13. (Cont'd)

recommended for validation. Issues covered relative to validation include how
much testing is required, i.e., termination criteria and what other concerns should
be addressed during validation.

15. (Cont'd)

Software design, Software testing, High integrity computing

DISCLAIMER NOTICE

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

LIST OF FIGURES

Figure No. and Title   **Page**
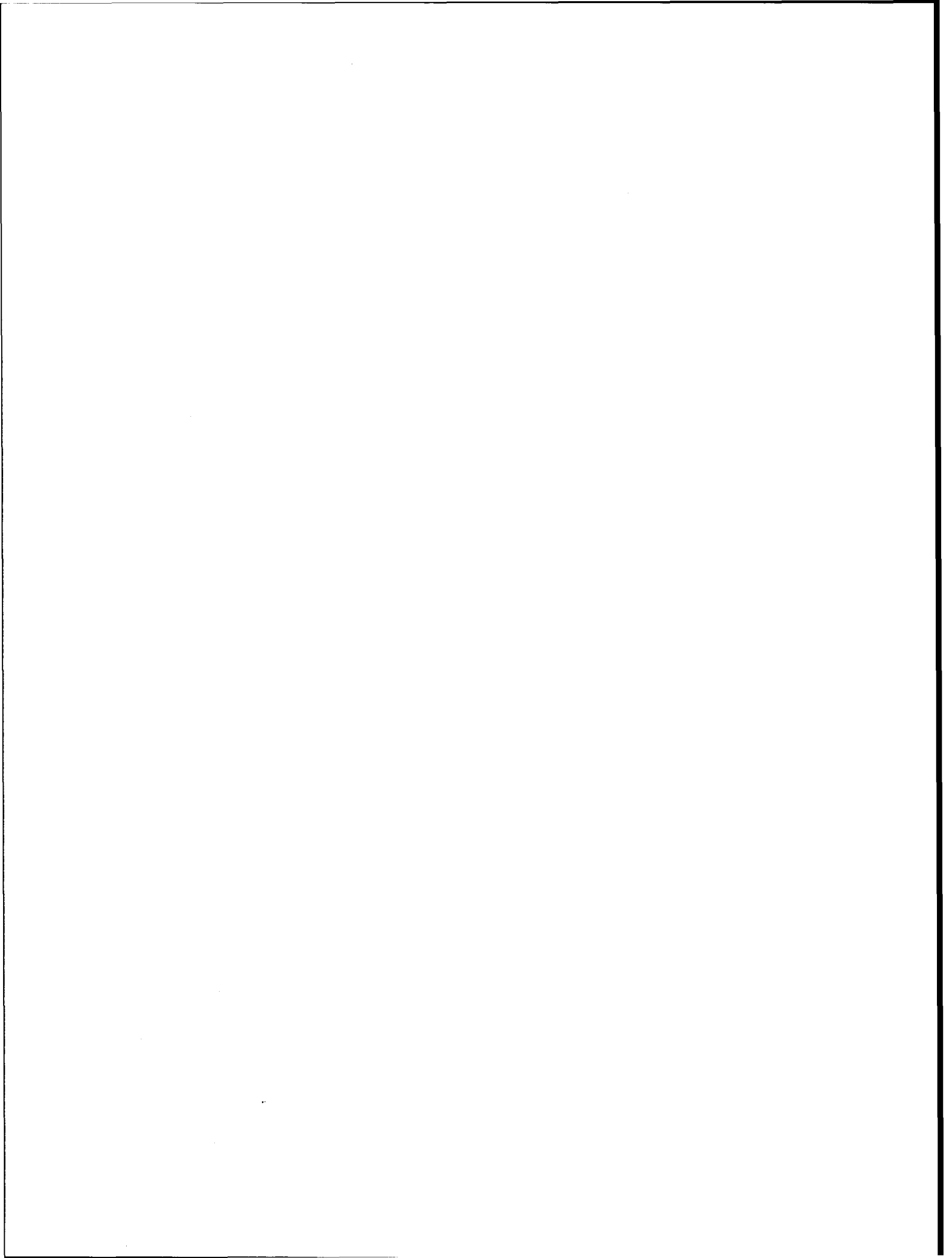
APPENDIX A

RISK BASED CLASSIFICATION GUIDELINES

The following material is furnished as an experimental guide for the use of risk based classification for nuclear plant protection systems. As shown in Sections 2 and 3 of this report, safety classifications for the nuclear field are application based (using the function served as the primary criterion), whereas those in use by the process industry and the military are risk based. There are obvious obstacles to the use of risk based classifications (and the associated integrity levels) for nuclear power plants, yet there are also many potential benefits, including

- it considers all capabilities provided for dealing with a specific hazard, thus assigning a lower risk where multiple protection is provided (either at the same or at lower layers); this permits the plant management to perform trade-offs between systems that meet the highest qualification levels or multiple diverse systems at lower qualification levels

- it motivates the use (and therefore also the development) of protection systems with demonstrated low failure probability

- it may permit lower cost process industry equipment of an established integrity level to be used in nuclear applications (subject to verification of the integrity level and regulatory approval)

The totality of these benefits may reduce the cost of digital protection systems significantly and motivate utilities to much more rapid upgrading of the capabilities than is currently the case. Therefore the outline of a risk based classification is presented here, to serve as a starting point for further investigation and possible trial application.

A.1 Format for a Risk Based Classification for Protection Systems

Risk has been defined as a function of the probability of an event and the severity of its consequences. The major obstacle to the use of risk based classifications in the nuclear field is the lack of data on the frequency of occurrence of failures, particularly of software failures. This can be alleviated by using software complexity as a surrogate for frequency of failure, the assumption being that a complex program is more likely to fail than a simpler one. Therefore two methodologies are presented: the primary one based on failure frequency, and an alternate one that uses software complexity instead.

Because only rough estimates of the frequency of failure are usually available, the following frequency definitions differentiate between groups that differ by at least a factor of ten. The following levels for expected frequency of failure can be used:

- *Frequent*   Expected value greater than 0.1 incidents per plant-year

- *Probable*   Expected value less than 0.1 but greater than 0.01 incidents per plant-year

- *Occasional*   Expected value less than 0.01 but greater than 0.001 incidents per plant-year

- *Remote*   Expected value of less than 0.001 but greater than 0.00001 incidents per plant-year

- *Improbable*   Expected value less than 0.00001 incidents per plant-year

Similarly, consequences are grouped into the following four levels:

- *Extreme*: Potential for multiple deaths in general population, e. g., a condition that requires declaration of a General Emergency without remaining in conditions of lesser emergency for at least 10 minutes

- *High*: Potential for multiple deaths in plant population, e. g., a condition that requires declaration of a site area emergency without remaining in a condition of lesser emergency for at least 10 minutes

- *Moderate:* A condition that requires declaration of an alert

- *Low:* Failure can give rise to an unusual event subject to notification requirements

Table A.1, which is largely derived from the U. K. MOD 0056 standard, shows the resulting risk assignment.

TABLE A.1. ASSIGNMENT OF RISK LEVELS

Failure Frequency	Severity			
	Extreme	High	Moderate	Low
Frequent	1	2	3	3
Probable	2	3	3	4
Occasional	3	3	4	4
Remote	3	4	4	4
Improbable	3	4	4	negligible

2

If it is desired to employ software complexity as a surrogate for expected frequency of failure the following risk classification is proposed:

TABLE A.2. MODIFIED ASSIGNMENT OF RISK LEVELS

Complexity	Severity			
	Extreme	High	Moderate	Low
High	1	2	3	3
Moderate	2	3	3	4
Low	3	3	4	4
Very Low	3	4	4	4

A noticeable change is that the five levels of expected frequency have been collapsed into four levels of complexity. This recognizes that use of a surrogate may lead to a loss of granularity; it is particularly difficult on the basis of complexity to distinguish between the bottom two levels of the frequency classification. In practice it is not believed that this difference will affect the classification significantly.

To make this classification useful for regulatory purposes guidance is provided for the assignment of consequence and frequency (or complexity) levels in Section A.3.

A.2 Classification of Implementation

The assessment and classification of systems based on implementation involves determining the extent of redundancy and the degree of independence. These issues are discussed in sections A.2.1 and A.2.2, respectively.

A.2.1 Classification Based on Degree of Redundancy

In a classification approach based upon the degree of redundancy, the analyst determines the number of failures *in the fault classes handled by the replicated system* that can be tolerated by the system prior to a degradation or cessation of service. Because redundancy is generally limited to four or fewer parallel channels, the classes into which systems are categorized in this approach can be unambiguously defined in terms of non-redundant, dual, triple, or quadruple redundancy.

"Systems" comprised of components which have varying degrees of redundancy are classified according to the components which have the least degree of redundancy. If classification by the redundancy of components yields an ambiguous result, then the system is classified by counting the least number of failures which are required to disable the system. A common problem with

3

this classification approach is that developers sometimes create systems which do not carry the redundancy through to a critical node, e. g., a system composed of redundant parallel functions that are evaluated in a single channel voting scheme. Considerable judgment is required to properly classify such a system.

Moreover, redundancy must be applicable to the fault classes that are a threat to the target system. For example, for the protection system the operation of fault tolerance provisions must be sufficiently prompt to prevent irreversible damage to the plant.

The degree of redundancy is an accepted indicator of the ability of the hardware components of systems to withstand multiple failures because of the implicit assumptions that failures are uncorrelated. This assumption cannot be generally substantiated for software and therefore other classifications are investigated below.

A.2.2 Classification Based on Degree of Independence

The number of successive faults to be tolerated is of much less importance in software than in hardware. Software does not undergo a physical change in connection with the failure, and therefore a failed program will continue to execute successfully as long as the data sets and computer states that led to the failure will not be repeated. But the degree of independence of back-up provisions is of primary importance because software failures are by definition due to design related faults that may also be present in alternates that have been supplied. Where the alternate is an exact copy of the failed software (e. g., a duplicate program running on another hardware platform) it is indeed certain that the fault will be present, but even under these circumstances the program may execute successfully because different data values and computer states will be encountered. For protective functions in nuclear power plants this possibility of successful execution cannot be counted on to provide a creditable back-up, and therefore the degree of independence must be evaluated.

The classification presented below is based on the following assumptions or premises:

(i)   Functional diversity provides a much greater degree of independence than software diversity for the following reasons:

> The specifications for the functionally diverse approaches are necessarily different, and care should be taken to provide true independence of specifications (by having them prepared by different organizations or at least individuals, avoidance of common source documents, etc.)

> The algorithms are different, provided only that dissimilar sensed data are used (and this should be an explicit requirement)

(ii) There may be practical considerations that make it highly desirable to run the functionally diverse software on computers of the same type (each version running on a separate

4

multi-channel computer system). This is not considered an impairment of independence if the computers have a history of reliable operation in the proposed environment.

(iii) Running functionally diverse software on the same set of multi-channel computers impairs independence, but the impairment can be minimized by avoidance of shared services and use of a simple proven scheduling algorithm and program which preclude the possibility of a single programming fault affecting both versions. If these steps are taken, functionally diverse programs running on the same computer are considered preferable to software diverse programs running on separate computers because the area of possible non-independence is smaller.

The following four-level classification of implementations is intended to be matched with the four-level risk classification for requirements:

1 - Functionally diverse programs resident on separate fault tolerant computers;

2 - Functionally diverse programs resident on the same fault tolerant computers[1];

3 - Software diverse programs running on the same (or separate) fault tolerant computers[1];

4 - Software employing fault avoidance resident on the same fault tolerant computers[1];

For environments classified as having negligible risk (see Table 1) software that meets the standards of good commercial practice will be acceptable.

Where extensive use of fault avoidance is demonstrated, involving both software development process control and formal verification methods, the implementation classification is relaxed by one step (increased numerically), except for risk 1 applications.

A.3 System Level Severity Classification

In assigning an automatic protection system or system function to the classification proposed above it must be kept in mind that the severity of the consequence is that of the automated portion (or function) alone. Thus automated protection for plant conditions which progress less rapidly can be assigned to a lower consequence category than those for plant conditions that require a rapid response. Thus, the automated initiation of engineered safety functions (ESF) is assigned a lower severity category than those for reactor trip.

---

[1] This means that the same two (or more) functionally diverse programs run on each of the (typically three or four) computers used for a given plant function.

A suggested general guideline (to be modified for the specific plant environment) for the severity assignment is:

Reactor trip system (sole automated trip initiation) - Extreme

Engineered safety function (sole automated function) - High

Reactor trip system (backed up) - Moderate

Functions that reduce the frequency of challenges - Moderate

All others - Low

A.3.1 System Level Frequency or Complexity Classification

The use of alpha and beta factors (see Section 2.1.1) will be helpful in arriving at the appropriate frequency ranges if that option of determining risk is selected. It may be known that a software product similar to one proposed for a safety system is running as part of a control system at 15 plants in the U. S., and that among these there has been an average of one discrepancy per year reported in the past several years. All discrepancies have been of a minor nature, resulting in either the output being somewhat delayed or being just outside the specified tolerances. It is concluded that the condition probability of a catastrophic failure (given that a discrepancy is detected) is less than 0.1. Since the probability of a discrepancy being detected is less than 0.1 per year, and the probability of a catastrophic failure, given that there is a discrepancy, is also less than 0.1, it is concluded that the expected failure frequency should be assigned to the *occasional* category (between 0.001 and 0.01 per year).

When two programs are arranged side-by-side most software professionals can probably decide which is the more complex one, but an objective definition of complexity is also difficult to come by (see the report on Quality Metrics, Document H0009, generated as part of this effort). For use by a regulatory agency the complexity measure should be simple to evaluate and not be tied to a proprietary tool. The information flow metric described by Henry and Kafura meets these requirements because it is based solely on the number of incoming and outgoing signals [HENR81]. The information flow has been used to express complexity in the example discussed below, and it was found that systems could be fairly easily classified by this criterion. It is expected that the assessment of complexity will benefit in some cases by also considering the number of operating modes of the computer based system, such as calibration mode, standby mode, operating mode and maintenance mode. An investigation of the benefits that will be achieved by the additional effort required to incorporate operating modes is still incomplete.

A.3.2 Classification by Software Function

Classification by software function can be used to reduce the risk level for code that does not directly affect the primary plant service. A fundamental requirement for use of reduced risk

levels is that the code associated with the lower risk function must not in any way interfere with the execution of the higher risk functions. Minimum safeguards are:

- the execution time of the lower risk functions is limited to a small fraction of the total execution time by trusted watchdog timers; in some cases, such as maintenance code, the execution of the low risk function is inhibited during normal plant operation.

- lower risk functions are prevented by a trusted mechanism from writing into the address space of the high risk functions; they may be permitted to read this address space.

An example of classification by software function is presented below.

a. Plant Safety Function: Software involved in the generation and outputting of commands used controlling safety actuators. This is equivalent to the command function as defined by IEEE-STD-603-1990;

b. Communication: Software involved in the transfer of data from its origination at a control function or transducer to its delivery to the user or process. Communications software works in conjunction with system and executive software in the transfer of data to and from processors;

c. Data Acquisition: Software involved in the processing, storage, and reduction of data after arriving at its destination process;

d. System and Executive: Software involving the operating system, vendor-supplied device drivers, low level processing to and from peripherals, and related functions;

e. User Interface: Software involved in the presentation of data to operators and other users either interactively or in the form of fixed displays. In the case of interactive applications this classification applies to the user interface software in combination with the system software;

f. Calibration: Software involved in setting constants and other aspects of calibration of sensors;

g. Surveillance: Software involved in the online monitoring and diagnostics of the functional status of I&C systems.

The identification of these functions can be derived from software and system design documentation produced by the digital systems developer. In general, functions a. and b. will be at the highest risk level (the risk level set according to Section 5.1). Functions c. and d. may in some installations be at a lower risk level, e. g., where a given data acquisition function affects only a single instrument out of a multiple set, or where a given executive function has been

7

widely used and found to be fault free over an adequate period of time. Functions e. through g.will normally be at least one level lower in risk than a. and b. at most installations.

The classification by software function can be used at the option of the licensee or developer. A typical benefit will be that in safety systems that are classified as risk level 3 the requirement for software diversity can be relaxed for sections of the code that carry a lower risk rating than the top groups, thus lowering the cost of implementation.

APPENDIX B

MEASUREMENT-BASED DEPENDABILITY EVALUATION

B.1. Introduction

While outage frequency and down time data on nuclear plants are freely available, data on software failures in nuclear plants are usually not disseminated. Yet assumptions about software failure modes and failure rate are essential for the assessment of verification and validation methodologies and for many other decisions that affect the evaluation of software used in safety systems for nuclear power plants.

In the course of our work on the Verification and Validation Guidelines we have drawn on software failure data from other applications, such as the NASA Space Shuttle and the FAA Advanced Automation System, to make up for the lack of data from the nuclear power environment. While inferences drawn from these data are considerably better than blind guesses, it is realized that they are not conclusive because:

 -the applications served by the software are different from those prevailing in safety systems (many are much more complex)

- the user environment is different (there are many more operating modes, and software experts are available on short notice when problems arise)

- they are produced by much larger software teams, using sophisticated tools and development methodologies, and frequently working in higher order languages than are currently in use for nuclear safety software.

It was therefore considered very desirable to undertake an analysis of a limited set of hardware and software failure data that became available as a result of the undertaking by the Tennessee Valley Authority (TVA) to report on failures of the Eagle 21 installation at the Sequoyah 1 and 2 plants for a period of approximately two years [TVA90]. The analysis which is described in the body of this report is intended to demonstrate that

- fairly unstructured data (semi-annual summaries of the failures were furnished in letter form, usually describing the event and then the action taken) can be summarized to present an easily assimilated picture of the overall reliability and of trends (Tables B-1 and B-2 of this report) and by origin of the failure (Tables B-3 and B-4)

- even a relatively small quantity of data can yield meaningful confidence intervals about the expected value of the reliability parameter (particularly in Table B-1)

- tools are available to propagate reported (usually component level) failure data to

9

meaningful reliability and availability estimates for the multi-channel system (Section 3). The assumptions stated in that section are conservative and may overstate the system unavailability. The availability of additional data (longer period of time from these installations or from other installations) will permit use of more realistic values of key parameters.

This report is not intended to indicate concern with the dependability or safety of the Eagle 21 installations at Seqouyah. Such an assessment is outside the scope of our work, and it requires evaluation of the modes and rate of demand on the safety functions, and also equivalent failure data for other digital installations as well as for analog ones. But it appears desirable to lay a foundation for measurement based assessment of digital safety systems, and it is hoped that this report will be a step in that direction.

The measurement based approach provides a more direct (and probably more objective) evaluation of the reliability and availability of protection systems than methodologies based on product or process metrics. To exploit this promising approach it is recommended that a commitment to furnish failure data be obtained as a part of future licensing of digital installations, and that the format for such data follow that recommended in the "Error Classification Guideline" generated as part of our effort.

B.2. Analysis of Failure Data

The analysis methods used in this section include: estimation of Mean Time Between Failures (MTBF), reliability growth analysis, and failure classification.

B.2.1. Overview of the EAGLE 21 System and Data

The EAGLE 21 system is a modular, microprocessor based process protection system developed by Westinghouse Electric Corporation [WEST91]. A typical system configuration has four redundant channels which monitor plant parameters to maintain safe operation conditions and initiate the actuation of reactor trip and engineering safeguard systems on the 2-out-of-4 vote. The EAGLE 21 system consists of five physically and electrically separate subsystems: Input/Output, Loop Processor, Tester, Man-Machine Interface (MMI), and Power Supply and Distribution.

The I/O Subsystem provides interface between process sensors and the Loop Processor Subsystem (LPS), and between the LPS and control systems. The Loop Processor Subsystem performs all functions necessary to accomplish the basic protection function, such as scanning and calibration of input signals, calculations, and generation of output signals. The Tester Subsystem, in conjection with the Man-Machine Interface cart, provides the means for testing system functions, verifying and adjusting setpoints and tuning constants, and performing automatic surveillance testing and other maintenance operations. The Power Supply and Distribution Subsystem provides separate DC power sources to different subsystems to prevent failure interaction between subsystems.

The failure data was collected from the Sequoyah Unit 1 and Unit 2 for a 2-year period. For the first half year, only Unit 1 was in operation. For the last half year, only the Unit 2 data was available. Thus, the data contains totally about 3 system years of failure reports. By a failure, we mean an abnormal or error condition detected either by the system itself or by human being. This definition may not be consistent with that proposed in [LAPR85]. Here, for the convenience of discussion, we do not distinguish between error and failure. A failure can be permenant or transient in nature, and can be caused by hardware or software problems. The severity of a failure can be divided into the component, channel, and system levels. In addition, failures can be detected during normal operation or during maintenance, which we call detection mode.

B.2.2. Estimation of MTBF

Assume that the random variable TBF is exponentially distributed with a mean of M(MTBF). If n failures have been observed during the measurement period T, then

$$\overline{X} = \frac{T}{n} \qquad (1)$$

is an estimate of M. Further, we can estimate a $100(1-\alpha)\%$ confidence interval of M. It can be shown that $2nX/M$ has a chi-square distribution with $2n$ degrees of freedom [TRIV82]. We can find two numbers, $x^2_{\alpha/2;2n}$ and $x^2_{1-\alpha/2;2n}$, such that

$$P(x^2_{\alpha/2;2n} < 2n\overline{X}/M < x^2_{1-\alpha/2;2n}) = 1-\alpha \qquad (2)$$

Thus, a $100(1-\alpha)\%$ confidence interval for M is

$$\frac{2n\overline{X}}{x^2_{1-\alpha/2;2n}} < M < \frac{2n\overline{X}}{x^2_{\alpha/2;2n}} \qquad (3)$$

In the following tables, for each estimated MTBF, a 90% confidence interval is provided in the adjacent paratheses.

B.2.3. Statistics by Detection Mode

Table B-1 shows failure statistics by detection mode, i.e., the system operation mode under which the failure is detected. Most failures (70%) were detected during system maintenance and had no effects on normal operation. For the failures occurring during normal operation, none of them disabled the system safety function, although some of them created half-trip conditions. No inadvertent actuation of reactor trip was actually created under these faulty conditions because other redundant channels were functioning properly. These results indicate the important roles of regular maintenance and channel redundancy in the EAGLE 21 system.

Table B-1. Failure Statistics by Detection Mode

Detection Mode	Frequency	Percentage	MTBF (days)
All	71	100	15.9 (13.2, 19.6)
Normal Operation	21	29.6	53.8 (38.9, 80.3)
Maintenance	50	70.4	22.6 (18.2, 29.0)

B.2.4. Reliability Growth Analysis

Table B-2 lists Mean Time Between Failures (MTBF) for the four subperiods in the whole data collection period. Each subperiod is approximately a half year. The calculation of the MTBF takes the average on the two measured units (systems). Included in parentheses are the 90% confidence intervals. There is no statistical difference between periods 1 and 2, and between periods 3 and 4, because the MTBF of both periods (1 and 2, or 3 and 4) falls into the other's confidence interval. However, reliability growth is obvious from the first year to the second year. This is true for both normal operation and maintenance modes.

Table B-2. MTBF by Subsystem (Half Year)

Detection Mode	Subperiod 1	Subperiod 2	Subperiod 3	Subperiod 4
All	11.3 (8.2, 16.7)	9.3 (7.2, 12.4)	55.4 (32.8, 118)	38.5 (19.9, 113)
Normal Operation	35.6 (21.0, 75.8)	29.3 (19.3, 50.8)	388 (130, 7564)	154 (51.4, 3002)
Maintenance	16.6 (11.4, 26.9)	13.5 (10.1, 19.3)	64.7 (36.9, 148)	51.3 (24.5, 188)

B.2.5. Failure Classification

Table B-3 classifies failures in terms of subsystems where they occurred. In addition to the five categories corresponding to the five subsystems in EAGLE 21, the category "other" includes events that do not fall into any of the five subsystems, such as unknown and operation failures. In the table, the percentage of failures occurring in each subsystem and the associated 90% confidence interval are provided. It is seen that nearly half failures occurred in the Tester subsystem. The number of failures occurring in the I/O, Loop Processor, and Power Supply subsystems are not statistically different, because each percentage midpoint falls into the other

12

two's confidence intervals.
Table B-3. Failure Distribution by Subsystem

Subsystem	Tester	I/O	Loop Processor	Power Supply	MMI	Other
Frequency	34	13	11	9	1	3
Percentage	47.9±11.6	18.3±9.0	15.5±8.4	12.7±7.7	1.4±2.4	4.2±4.7

Table B-4 classifies failures by type. The software MTBF is much better than the hardware MTBF. The two software specification errors involve equations for calculating the overtemperature, overpower, and comparator trip setpoints. Thus, although none of the reported software failures occurred during the normal operation of the system, they could have serious impact on the safety function.

The most significant dependability impediment is the number of processor failures. All these failures were processor lockups due to the time characteristics of the Intel 82284 clock generator and were later eliminated by replacing the Intel clock generator with a Siemens model. Multiple channel failures caused by the same fault (common mode failures), or multiple channel failures occurring coincidentally, are not found in the data.

Table B-4. Failure Statistics by Type

Category	Type	Frequency	Percentage	MTBF (days)
Hardware	Processor	37		
	I/O	10		
	Power	9	63	88.7	17.9 (14.8, 22.4)
	Bus	5		
	Memory	1		
	Unknown	1		
Software	Specification	2		
	Initialization	1	4	5.6	282.5 (145.7, 827.0)
	I/O Driver	1		
Skinware	Installation	2	4	5.6	282.5 (145.7, 827.0)
	Operation	2		

B.3. Dependability Modeling

A Markov model can be built for the EAGLE 21 system to evaluate system availability. To show the impact of hardware failures and software failures respectively, we first develop a model under only hardware failure conditions and then add software failure conditions to the model.

B.3.1. Availability without Software Failures

Figure B-1 shows a Markov model for the EAGLE 21 system subject to only hardware failures. The model assumes that there are four identical channels and all of them have the same failure and repair rates. Following is the notation used in the model:

N normal state in which the system is able to provide the safety function
C_i channel failure state in which i channels have failed
M maintenance state in which one channel is being maintained
F failure state in which the system is unable to provide the safety function
λ_c channel failure rate
λ_m maintenance rate
μ channel repair rate
μ_m reciprocal of the average maintence time

Since the modeled system is a 2-out-of-4 system, if not at least 2 channels are available, then the system will fail. Thus, the occupancy probability of state F, P_f, is the system unavailability, and $1-P_f$ is the system availability.
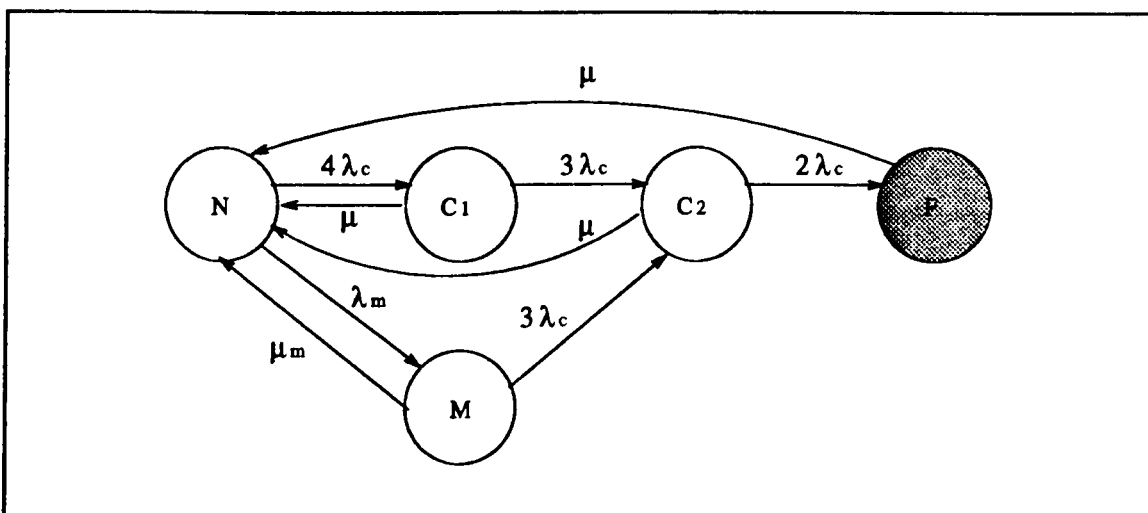
Figure B-1. The EAGLE 21 System Model without Software Failures

To estimate unavailability, the above parameters are estimated as the following values based on the data or on experience:

$\lambda_c = 1.9 \times 10^{-4}$/hour (measured from the data)

$\mu = 0.5$/hour (2 hours of repair time)

$\lambda_m = 1.4 \times 10^{-3}$/hour (scheduled mainentance once a month)

$\mu_m = 0.25$/hour (4 hours of duration for each scheduled mainentance)

By substituting these parameter values into the model and solving it using SHARPE [SAHN87], the unavailability is evaluated as 6.33×10^{-9}.

B.3.2. Availability under Software Failures

Secondly, we take software failures into account. Figure B-2 shows a modified Markov model (to Figure 1) which includes software failure conditions. The model assumes that hardware failure and software failure are independent of each other. But software failures in different channels are related. If a software failure is deterministic, i.e., it does not depend on timing and system state, all channel's software may fail together. If a software failure is nondeterministic, i.e., it is timing and system state related, the software failure may occur only in one channel. Nondeterministic software failures have been found to be very significant in real-time systems [HECH93, LEE93]. Following is the additional notation used in the modified model:

S_i software failure state in which software in i channels have failed

λ_s failure rate of software in a channel

λ_{s2} failure rate of software in the second channel after the first software failure

λ_{s3} failure rate of software in the third channel after the second software failure
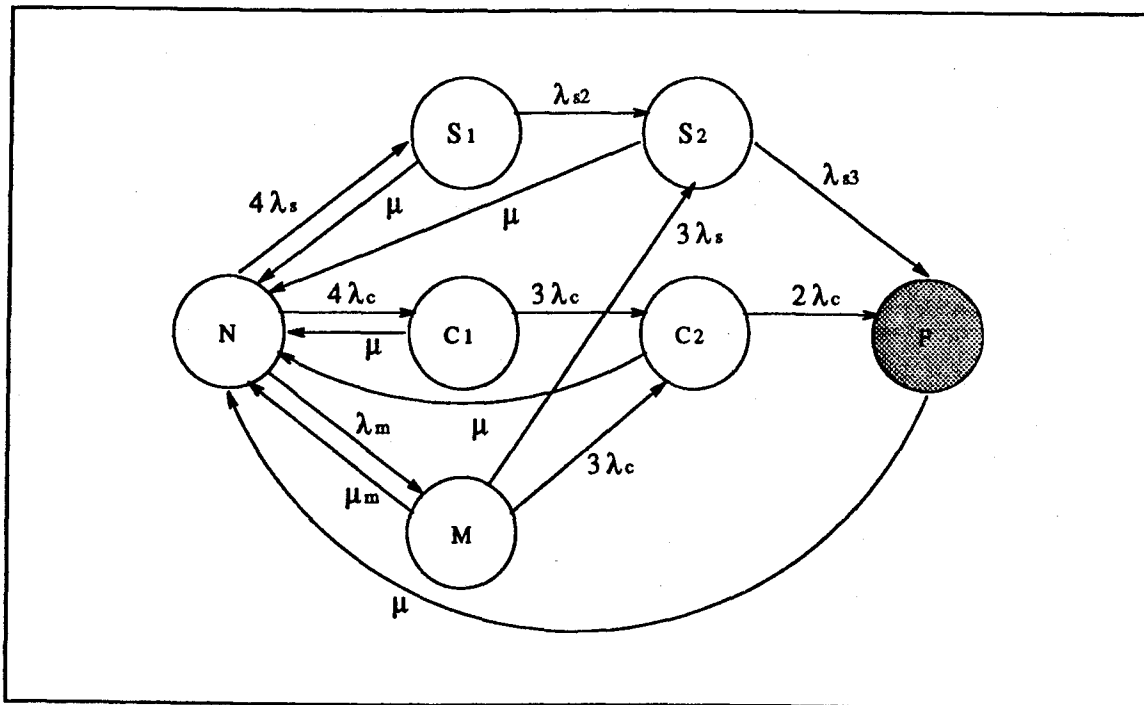
Figure B-2. The EAGLE21 System Model under Software Failure Condition

15

Since software failures are related, λ_{s2} is not simply $3\lambda_s$ and λ_{s3} is not simply $2\lambda_s$, as we have done for hardware failures. In order to determine λ_{s2} and λ_{s3}, correlation parameters have to be introduced. A validated correlation model, called p-dependent model, has been proposed in [TANG92]. In the model, a correlation parameter, $p_{i,i+1}$, is defined as the probability that one more component will fail, given that i components have failed. The paprameter is then used to calculate the rate from state i (with i failed components) to state $i+1$ (with $i+1$ failed component), $\lambda_{i,i+1}$, by the equation

$$\lambda_{i,j+1} = \frac{p_{i,j+1}}{1-p_{i,j+1}}\mu_{i,j-1} \tag{4}$$

where $\mu_{i,i+1}$ is the recovery rate in state i.

In our case, $p_{i,i+1}$ represents the probability that the $(i+1)$th software copy will fail given that the ith software copy has failed. Therefore, λ_{s2} and λ_{s3} can be estimated by

$$\lambda_{s(i+1)} = \frac{p_{i,j+1}}{1-p_{i,j+1}}\mu \tag{5}$$

where i equals 1 or 2, and μ is the software recovery rate which is assumed to be the same as the channel repair rate. If we assume $p_{12}=p_{23}=0.5$, i.e., there is a 50% chance that another software copy will fail after each failure of a software copy, then λ_{s2} and λ_{s3} are estimated as 0.5/hour (same as μ).

B.3.3. Estimation of λ_s

Because no software failure was observed during normal operation for the measured period, the average software failure rate, λ_s, cannot be estimated from the data. However, if we assume that the software TBF (Time Between Failure) is exponentially distributed, an upper bound of λ_s can be estimated from the data.

Let T denote the measured time period in which no software failure has been observed. Assume that λ_0 is an upper bound of λ_s. We make hypotheses:

H_0: $\lambda_s \geq \lambda_0$
H_1: $\lambda_s < \lambda_0$

Now we use a statistical test to reject H_0 at a confidence level of $1-\alpha$. We know that

$$F(t) = 1-e^{-\lambda t} \tag{6}$$

is the c.d.f. of the software TBF. Thus, if H_0 holds, the probability that TBF is less than or equal to T should satisfy

$$P(TTE \leq T) = F(T) \geq 1 - e^{-\lambda_0 T} \qquad (7)$$

$$1 - e^{-\lambda_0 T} = 1 - \alpha \qquad (8)$$

Make this probability equal to $1-\alpha$:
Since we have not seen a single TBF instance within T, we should reject H_0 at the confidence level of $1-\alpha$, or at the significance level of α. By solving the above equation, we obtain

$$\lambda_0 = \frac{-\ln\alpha}{T} \qquad (9)$$

which is an upper bound of λ_s, although it may be too conservative.

By applying the above method to the data, an upper bound of λ_s is estimated as 8.5×10^{-5}/hour with a 90% confidence. Plugging this number into the model in Figure 2, we get a conservative estimate of the system unavailability 1.70×10^{-4}. Note that the estimated software failure rate (10^{-5}) is lower than the estimated hardware failure rate (10^{-4}). However, the system unavailability is significantly degraded by software failures. This is because there is no software fault tolerance in the modeled system.

B.4. Conclusions

In this paper, we demonstrated an analysis and modeling methodology in dependability evaluation for critical systems. Based on the EAGLE 21 failure data, a system dependability model was developed. The model evaluation showed:

1. The system unavailability caused by hardware failures does not exceed 10^{-9}.
2. Under a conservative estimation of software failure rate, the system unavailability degrades to 10^{-4}.

The estimation of software failure rate, nondeterministic software failures and the associated correlation parameters were also discussed in the paper.

REFERENCES

HECH93   M. Hecht, P. Crane, G. Dinsmore, and P. Thambidurai, *Analysis of CED PTRs*, Informal Technical Report, SoHaR Inc., May 1993.

LAPR85   J.C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," *Proc. 15th Int. Symp. Fault-Tolerant Computing*, pp.2-11, June 1985.

LEE93   I. Lee and R.K. Iyer, "Faults, Symptoms, and Software Fault Tolerance in the Tandem GUARDIAN90 Operating System," *Proc. 23rd Int. Symp. Fault-Tolerant Computing*, June 1993.

SAHN87   R.A. Sahner and K.S. Trivedi, "Reliability Modeling Using SHARPE," *IEEE Trans. Reliability*, Vol. 36, No. 2, pp. 186-193, June 1987.

TANG92   D. Tang and R.K. Iyer, "Analysis and Modeling of Correlated Failures in Multicomputer Systems," *IEEE Trans. Computers*, Vol. 41, No. 5, pp. 567-577, May 1992.

TRIV82   K.S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

TVA90   TVA letter to NRC dated May 10, 1990, "Sequoiyah Nuclear Plant (SQN) - Eagle 21 Functional Upgrade Commitments", and NRC letter to TVA dated October 31, 1990, "Reactor Protection System Upgrades and Enhancements - Sequoyah Nuclear Plant, Unit 2"

WEST91   *EAGLE 21 Technical Description*, Westinghouse Electric Corporation, Process Control Division, Pittsburgh, PA, Jan. 1991.

APPENDIX C

USE OF THE ECT TOOL FOR VERIFICATION

C.1   Motivation

The ECT (the acronym stands for Enhanced Condition Table) tool automatically extracts the implemented condition tables from source code. If the specification is in condition table format, the implemented tables and the required tables can be compared to determine exact correspondence between the specification and the implementation. The use of condition tables as a precise statement of requirements for nuclear safety software has been advocated by D. L. Parnas and is now being practiced at Ontario Hydro. The benefits of a precise statement of requirements is seen in the following example for the reactor feedwater control system (RFWCS) for a U. S. nuclear plant (the RFWCS is not a Class 1E system). Figure C.1 is a reproduction of the requirements for the water level control portion of the system; the ambiguities and inconsistencies are commented on below and it is then shown how conversion to an input/process/output (IPO) format and subsequent statement of the processes as condition tables surfaces and resolves these difficulties and gives insight into previously unrecognized modes of system operation.

C.2   The Feedwater Control Example

4.2.1   Inputs

    4.2.1.1 Reactor Vessel Level

        The RFWCS shall utilize the existing three narrow range
        level loops, L-3-53, L-3-60, and L-3-206, which will be input
        and validated. Any vessel level signal greater than ±5% of
        the average shall be considered invalid. When there are
        only two valid signals, an alarm shall be generated for a
        deviation of 10% between the two valid signals. The
        average of the three level signals will be utilized to produce
        a compensated level signal for control function input and
        level indication. Upon loss of validation for one level
        signal, the average of the two remaining signals will be
        utilized. Upon loss of validation for two level signals, the
        remaining valid signal will be used. Upon loss of all valid
        level signals, the RFWCS shall generate an alarm and
        revert to manual control.

Figure C.1   Excerpt from RFWCS Specification

The importance of the IPO format lies in its isolation of the process. The objective of the requirements analysis part of verification is essentially to determine whether the process can

handle the inputs resulting from all foreseeable conditions. Requirements which do not clearly isolate the process make the analysis at best difficult and at worst inconclusive, as can be seen in the RFWCS specification. The process of validation is implied in the sentence "Any vessel level signal greater than ±5% of the average shall be considered invalid." Incorrect or ambiguous expressions that are used in this sentence will be commented on below. For the present it is interpreted as "Any vessel signal not within ±5% of the average of the narrow range signals shall be considered invalid." Now assume that the narrow range extends from 0 to 100 and that the current correct reading is 60. One of the sensors fails and reads 0, while the other two continue to read 60. The instantaneous average reading is 120/3 = 40, and 5% range is 38 to 42. All signals are now invalid -- hardly the intended result.

This motivates a second interpretation which changes the key sentence to "Any vessel signal not within ±5% of the average of the narrow range signals during the preceding cycle shall be considered invalid." Let the signals during the preceding cycle be 58, 60, and 62, and let the highest one fail to zero. The remaining two signals are within ±5% of the average and can be validated. However, if the previous readings had been at the low end of the narrow range, say at an average of 5, a small increase in the level (e. g., to 5.3) could have simultaneously invalidated all. Any statement of tolerances purely as a percentage of the prevailing average carries with it a risk of unintended rejection at the low end of the range. It is possible that this problem may not arise due to limitations on the increase in level between computing cycles, but this will have to be investigated in further analysis.

In the first restatement of the sentence we substituted " ... not within ±5% of the average ..." for the original " ... greater than ±5% of the average ...". It is assumed that the original wording represented an editing oversight. If it did not, then further clarification will be required and our interpretation will have to be revised. The other correction that was made at the same time was to substitute "... average of the narrow range signals ... " for " ... average ...". The reason for that substitution was to remove the possibility of using the average total water level as a criterion. If our interpretation is incorrect, the restatement of the process will of course have to be revised. But the "average" without qualification used in the original statement is obviously ambiguous and represents an obstacle to verification, regardless of the technique that is used.

Another process, this one more explicitly defined by the use of "shall" is introduced in the sentence that follows the one discussed above: "When there are only two valid signals, an alarm shall be generated for a deviation of 10% between the two valid signals." Here, again, the percentage needs to be referenced to the narrow range signals. Further, clarification is required whether the threshold is 10% of the lower or higher of the two signals, or whether it is 10% of their average. The action " ... alarm shall be generated ..." can mean a specific alarm, a general alarm without further qualification, or a general alarm with a subsidiary indication of its origin.

Two additional processes are required by par. 4.2.1.1 of the specification: generation of the compensated level signal (under three different conditions of signal validation), and the generation of an alarm and reversion to manual control when all valid signals are lost. In the first of these requirements there is uncertainty about how a single valid signal shall be identified.

The preceding process required generation of an alarm when there was a significant deviation between two remaining valid signals, but no guidance is provided about how one of these can be rejected and the other one declared valid. Possibly this is an operator decision, but in any case the conditions for arriving at a single valid signal need to be identified. Similarly, the conditions for declaring a single valid signal invalid have not been identified. The alarm mentioned in the last process needs the same clarification that was discussed above.

Thus, par. 4.2.1.1 has been analyzed as containing six processes which will be identified by a dash (-) number attached to the paragraph designation. The following description of these processes are only summaries that tie them to the previous discussion. The methodology for specifying the process will be discussed later on.

4.2.1.1-1   The three narrow range level signals shall be validated

4.2.1.1-2   If there are only two valid signals, an alarm shall be generated when they differ by more than 10%

4.2.1.1-3   (By a process to be identified, and following the alarm) one of the two signals shall be declared valid

4.2.1.1-4   (By a process to be identified) the remaining valid signal may be declared invalid

4.2.1.1-5   A compensated level signal shall be generated as input to the control function and as level indication. The conditions for generating this signal depend on the number of valid narrow range level signals available.

4.2.1.1-6   If all valid signals are lost an alarm shall be generated.

The decomposition of the loosely drawn requirements of the specification into simple processes with much more explicit action requirements permits other analysis techniques to be applied. One example is Failure Modes and Effects Analysis (FMEA). The conventional starting point for hardware FMEA is a parts list, because (a) failures arise in parts, and (b) the parts list can claim to be exhaustive. Thus an FMEA based on a parts list has a solid ring to it. The equivalent of the parts list for software is the list of processes. Failures arise in processes, and the list of processes can be claimed to be exhaustive (this latter claim can not be made with respect to "functions").

C.3   Process Specification

The definition of the -1 process identified above was "The three narrow range signals shall be validated". If the current state (as determined by validation at the previous cycle) starts with three valid signals, then validation will be based on the negative requirement developed previously: "Any vessel signal not within ±5% of the average of the narrow range signals during the preceding cycle shall be considered invalid." To be valid, the signal must be within ±5% of

21

the average of the preceding cycle. A simple mathematical notation is introduced which is defined below.

sn (n = 1, 2, 3): current cycle narrow range sensor signals

Sn (n = 1, 2, 3): current cycle validated sensor signals

S'n (n = 1, 2, 3): previous cycle validated sensor signals

$S'' \equiv (S'1 + S'2 + S'3)/3$ (average previous cycle signal)

For three previous cycle valid signals the conditions for current cycle valid signals can then be defined as

$$Sn = \begin{cases} sn & \text{for } .95\,S'' \leq sn \leq 1.05\,S'' \\ \varnothing & \text{otherwise (a member of the null set)} \end{cases}$$

These relations are in more detail expressed in the following condition table. In the upper part of the table the first column lists the conditions applicable to a row, and the entries in the other columns indicate whether the condition is true (y) or false (n). In this format the columns are also referred to as *rules*. The lower half of the table lists the action (assignments) that are desired for the rule in that column.

Table C-1. Conditions for Three Previously Validated Signals

.95S"≤s1≤1.05S"	y	n	y	y	y	n	n	n
.95S"≤s2≤1.05S"	y	y	n	y	n	y	n	n
.95S"≤s3≤1.05S"	y	y	y	n	n	n	y	n
	S1=s1	S1=∅	S1=s1	S1=s1	S1=s1	S1=∅	S1=∅	S1=∅
	S2=s2	S2=s2	S2=∅	S2=s2	S2=∅	S2=s2	S2=∅	S2=∅
	S3=s3	S3=s3	S3=s3	S3=∅	S3=∅	S3=∅	S3=s3	S3=∅

The table reveals that it is possible to transition from the normal (three valid signals) state to a state with only a single valid signal or possibly even to one with no valid signals. This, by itself, is a significant result of the use of this methodology. The table also provides a starting point for a very intensive test approach that is useful for validation but is outside the scope of the present discussion.

For only two valid signals during the preceding cycle, designated S'm (m = a, b), and the assumption that the 10% criterion applies to the average of the two signals, we define

$$S' \equiv (S'a + S'b)/2$$

$$Sm = \begin{cases} sm & \text{for } |sa - sb| \leq 0.1\ S' \\ \varnothing & \text{otherwise} \end{cases}$$

The condition table for this case is shown below. The following combinations are considered physically impossible and have been omitted: the difference between the signals is within the threshold, but each signal deviates by more than 5% from S'; and the difference between the signals is outside the threshold, and both signals deviate by less than 5% from S'.

Table C.2 Conditions for Two Previously Validated Signals

$|sa - sb| \leq 0.1 S'$	y	y	y	n	n	n
$|S' - sa| \leq .05 S'$	y	n	y	y	n	n
$|S' - sb| \leq .05 S'$	y	y	n	n	y	n
	Sa=sa	Sa=sa	Sa=sa	Sa=sa	Sa=$\varnothing$	Sa=$\varnothing$
	Sb=sb	Sb=sb	Sb=sb	Sb=$\varnothing$	Sb=sb	Sb=$\varnothing$

Here, again, considerable insight is gained through use of condition tables, e. g., in the third through fifth column of the table. The results shown in the bottom two rows may not be those anticipated by the user; if changes in the process are desired, the table provides an excellent medium for specifying them in a precise and complete manner.

For only a single valid signal during the preceding cycle, designated S'k, and the assumption that validation requires it to be within 5% of its previous value,

$$Sk = \begin{cases} sk & \text{for } |S'k - sk| \leq 0.05\ S'k \\ \varnothing & \text{otherwise} \end{cases}$$

The condition table for this case becomes simply

Table C.3 Conditions for One Previously Validated Signal

$|S'k - sk| \leq 0.05\ S'k$	y	n
	Sk = sk	Sk = $\varnothing$

Similar condition tables will be developed for every safety critical process.

23

C.4   Regeneration of Condition Tables

The work described in this subheading utilizes the ECT (enhanced condition table) tool that has been developed by SoHaR under contracts from NASA and the U. S. Navy [HECH90]. ECT was primarily developed to improve test coverage, and these capabilities will be utilized in the test methodology described in the following subheading. A substantial benefit for verification is that ECT generates the condition tables that are implemented in the code. These can then be compared with the tables that define the process, *thereby verifying that the design and coding correctly implement the requirements*, and greatly reducing the concern with the contractor's software development environment.

An example of a C program and the condition table automatically generated from it by ECT are shown in the following figure. The program is a translation into C of a simple text formatter described in the Goodenough and Gerhart paper that proposed the use of condition tables for test data selection [GOOD75]. In this program the exclamation mark (!) indicates negation, and a double vertical bar ($||$) denotes the inclusive OR. The rows and columns are interchanged from the conventional presentation used in the reference and in the preceding subheading to facilitate printing where many rules need to be evaluated. Also, some additional symbols are introduced: the dash (-) indicates a "don't care" condition (the output is not affected by the state of this condition), and (y) and (n) are used to denote conditions necessarily imposed by a preceding condition. As an example of the latter notation, consider conditions c1 and c2. If the current input character is a BL (blank line) it cannot at the same time be an ET (end of text code). Therefore the first four entries in the c2 column are (n).

```
main()                                          Condition Table (w/semantics
{                                               knowledge) for function "main"
 Alarm = false;
 bufpos = 0;                                    c1:  CW==BL
 fill = 0;                                      c2:  CW==ET
 do {                                           c3:  bufpos!=0
    incharacter (&CW);                          c4:  fill+bufpos<MAXPOS
    if (CW == BL || CW == ET) {                 c5:  fill!=0
       if (bufpos != 0) {                       c6:  bufpos<MAXPOS
          if (fill + bufpos < MAXPOS && fill != 0) {
             outcharacter (BL);
             fill = fill + 1;
          } else {                              c1   c2   c3  c4  c5  c6
             outcharacter(NL);                  y    (n)  y   y   y   (n)
             fill = 0;                          y    (n)  y   y   n   (n)
          }                                     y    (n)  y   n   -   -
       }                                        y    (n)  n   -   -   (n)
       for (k = 1; k<= bufpos; k++)             n    y    y   y   y   (n)
          outcharacter(buffer[k]);              n    y    y   y   n   (n)
       fill = fill + bufpos;                    n    y    y   n   -   -
       bufpos = 0;                              n    y    n   -   -   (n)
    } else {                                    n    n    (y) (n) -   y
       if (bufpos == MAXPOS)                    n    n    -   -   -   n
          Alarm = true;
       else {
          bufpos = bufpos + 1;
          buffer[bufpos] = CW;
       }
    }
 } while (!(Alarm || CW == ET));
}
```

Figure C.2 Condition table for text formatter

24

The condition table generated from the code exactly duplicates the one developed in the paper from the statement of the problem; the action listing (corresponding to the lower part of the tables shown in the preceding subheading) is suppressed in the figure. The recognition of conditions that necessarily result from preceding conditions is accomplished by the semantic analysis portion (SEM) of the ECT program. SEM generally identifies infeasible rules and thereby reduces both the volume of the condition tables and the amount of test cases that have to be investigated. An example of a semantically infeasible rule is represented by the following hypothetical timesheet program:

```
IF day_of_week = Sunday
      start new timesheet
Enter daily hours

...

IF day_of_week = Saturday
      total hours for week
```

Since the day of the week cannot be both Sunday and Saturday these conditions will never have to be evaluated together, and it should not be attempted to construct a test case in which both IF statements are true.

The ECT program currently operates on Ada and C programs. If a BNF (Normal Form) grammar exists for other languages used in the control system, then a parser can be for the ECT can be created by inputting the logic into the *yacc* UNIX-based software tool. For other languages (e.g., ladder logic and many assembly languages), the recreation of the condition tables can be performed manually, aided by static analysis tools, as is the current practice at Ontario Hydro.

The condition tables generated by ECT are compared with those produced as part of the process specification. In many cases there will be an exact match, indicating that the requirements have been correctly implemented in the design and coded into the program. In other cases differences may be noted, and these may be due to:

- redesignation of program variables (in particular to designate temporary assignments)

- re-ordering of predicate evaluation for programming efficiency

- undocumented communications between specifiers and designers, or between designers and programmers

- artifacts due to programming style

- misinterpretations of the requirements or the design.

The first two of these causes are to be expected. The other three are undesirable but do not necessarily result in an unacceptable program. In the review of the original CANDU/OH shutdown program most of the discrepancies of this type had at worst an insignificant effect on program operation. But as part of verification all of them will need to be investigated and documented, and comparison of the condition tables is a very efficient way of accomplishing this.

C.5 Conclusions

The use of ECT has the potential of overcoming many difficulties currently encountered in the verification and validation of high integrity systems. Some of these were pointed out in the body of this report:

- it targets differences between requirements and code, and hence is effective in detecting both unimplemented requirements and unintended functions (functions that do not implement requirements)

- most steps in the procedures required for its use in V&V can be automated, and hence the cost can be bounded

- it is an inherently independent activity, regardless of who performs it

- it does not require access to the developers design or development process, and it is therefore applicable to commercial and proprietary software

It is also effective in identifying abnormal conditions and events (ACEs) that can interfere with the normal execution of the code. An example of this is the simultaneous failure of two sensors for which no action was not specified in the textual requirements but for which appropriate handling was indicated in the condition table statement of the requirements (Table C-1).

Because of the large potential pay=off, further research in this technique is recommended.

REFERENCES

GOOD75   J.B. Goodenough and S.L. Gerhart, "Toward a Theory of Test Data Selection", *IEEE Transactions on Software Engineering*, Vol. SE-1, No.2, pp.156-173, June 1975

HECH90   M.Hecht, K.Tso, and S.Hochhauser, "Enhanced Condition Tables for Verification of Critical Software", *Proc. 7th International Conference on Software Testing*, San Francisco, June 1990.

APPENDIX D

THE USE OF CATS TO VERIFY EXECUTABLE CODE

The Code Analyzer Tool Set, CATS, was developed by Technicher Überwachungs Verein (TÜV) Norddeutschland for the purpose of analyzing (and subsequently certifying) embedded controllers for small microprocessor systems. TÜV is a quasi-governmental organization for certifying the safety and suitability of technology related products. The organization's name can be loosely translated as "Institute for Technical Protection". The tool set is intended to provide a detailed understanding of all operationally significant attributes of a program, typically one that is furnished for certification on a ROM (read-only memory) chip.

The major components of CATS are

- a disassembler -- DISCAT

- a program analyzer -- PROCAT

- a control flow analyzer -- COCAT

- a data flow analyzer -- DACAT

The functions and benefit for verification of each of these tools is presented below. A real-time routine developed at SoHaR was submitted for analysis by CATS, and the results of this are discussed in the last section of this appendix.

D.1   Disassembler -- DISCAT

Disassembly is the process of reconstructing assembly level code from machine code (executable code). The disassembly by DISCAT is path related, while in most other disassemblers it is governed by the memory structure. In path related disassembly the assembler level code will be generated in the order in which it is likely to be executed. In disassembly by memory structure the assembler code is generated in the order in which it is loaded in memory, which frequently means that code will be interspersed with data, and that the analyst will have to reconstruct jumps and calls.

The primary outputs of DISCAT are:

- a metrics file, providing overall program statistics (Fig. D.1)

- assembly level code (Fig. D.2)

- lists of exceptions and memory address ranges that were not disassembled (because they contained data or other non-code information)

- data files that are input to other CATS programs.

```
╔══════════════════════DISCAT 8086 - Vers.:1.1══════════════════════╗
║ ┌────────────────────────────────────────────────────────────────┐ ║
║ │ system-id      : HB                    date :13-05-1993   time :14:17:03 │
║ │ objectfile     : HB.HEX                file length (byte) : 25187 │
║ │ startadresses  : 0000:064B (*Rem 1*)                            │
║ │                                                                 │
║ │ result filename: HB                    translated bytes    : 10801 │
║ │ selected output: AMFD                  username: u            │
║ └────────────────────────────────────────────────────────────────┘ ║
║                                                                      ║
║ ┌─────────────────────────────results────────────────────────────┐ ║
║ │ number of programparts :    1                                   │
║ │ number of hw-interrupts:    0    number of tasks      :    0    │
║ │ number of subroutines  :   91                                   │
║ │ number of linear blocks:  992                                   │
║ │ number of instructions : 4737                                   │
║ │ number of jumps        :  192    number of indjumps    :    1   │
║ │ number of branches     :  486                                   │
║ │ number of calls        :  212    number of indcalls    :   24   │
║ │ number of indmoves      :  992                                   │
║ │                                                                 │
║ │ number of errors       :   34                                   │
║ └────────────────────────────────────────────────────────────────┘ ║
║                                                                      ║
║ Rem 1: Start address of module main.                                ║
╚══════════════════════════════════════════════════════════════════╝
```

Figure D.1 DISCAT Metrics

The metrics file can be used for verifying that the input file (or the ROM chip) did not contain extraneous material, and whether the machine code contained unusual constructs that might give problems in execution. The example shown in Figure D.1 lists 34 errors (more appropriately: exceptions) most of which were due to the use of an indirect address mode that is permissible in the processor used for this code but was not in the processor model used by CATS. Flagging of these exceptions is valuable when applications in a broader environment are anticipated.

A sample of the assembly level code generated by DISCAT is shown in D.2. The top part of the figure is a diagram of the overall routine structure, while the bottom part is a listing of the assembly instructions in the order in which they are expected to be encountered. If the compiler generates an assembly output, it can be directly compared with the disassembled code. In other cases the program structure of the source code can be compared with that generated in disassembly.

```
**********************************************************************
DISCAT51 - Vers.:1.2a                    date:25-09-1992  time:14:00:54
                              DEMO.ASM

                              page: 0
**********************************************************************
                            program list
                            ------------
        label     adr       instruction

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        prog0000  0000      ljmp 0036
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        inte0003  0003      ljmp 0003
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        inte000B  000B      ljmp 000B
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        inte0013  0013      ljmp 0013
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        inte001B  001B      ljmp 001B
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        inte0023  0023      ljmp 0023

        jump0036  0036      anl rs90,#F0
                  0039      setb bB3
                  003B      setb bB2
                  003D      setb bB5
                  003F      mov r00,#08
                  0041      mov r02,#78
                  0043      clr rE0

        jump0044  0044      mov (r00),rE0
                  0045      inc r00
                  0046      djnz r02,0044

        next0048  0048      mov r22,#2C
                  004B      mov r02,#00
                  004D      mov dptr,#8000
                  0050      clr rE0
                  0051      ljmp 0056

        jump0054  0054      inc r02
                  0055      inc dptr

        jump0056  0056      movx [dptr],rE0
                  0057      cjne r02,#FF,0054
```

Figure D.2 DISCAT Assembler Listing

29

D.2   Program Analyzer -- PROCAT

PROCAT is the program analyzer constituent of CATS. Its main purpose is to assist in top level analysis of program execution. The primary outputs are:

- an overview table (Figure D.3)

- a calling matrix (Figure D.4)

- a hierarchy file

The following pictures show output examples for some of the result files mentioned above.

overview table:

```
routine overview

routine typ
   |     address of entry block
   |       |     adress of lowest routine block if it is lower than entry
   |       |     | block
   |       |     |      address of last routine block
   |       |     |        |     number of blocks in the routine
   |       |     |        |       |    number of exits if different from one
   |       |     |        |       |       |   codesharing with other
   |       |     |        |       |       |   | routines
   |       |     |        |       |       |   |    number of direct cal-
   |       |     |        |       |       |   |    | led subroutines
   |       |     |        |       |       |   |    |    number of indi-
   |       |     |        |       |       |   |    |    | rect called
   |       |     |        |       |       |   |    |    | subroutines
   |       |     |        |       |       |   |    |    |   nesting
   |       |     |        |       |       |   |    |    |   | depth
   |       |     |        |       |       |   |    |    |   |   re-
   |       |     |        |       |       |   |    |    |   |   cursion
```

RTyp	EntryBl	FirstBl	LastBl	BlockNb	ExitNb	CShar.	DSR	OSR	Nest	Rec
Prog	000000		000095	8	NoExit		18	24	4	
Int	000003		000003	1	NoExit					
Int	00000B		00000B	1	NoExit					
Int	000013		000013	1	NoExit					
Int	00001B		00001B	1	NoExit					
Int	000023		000023	1	NoExit					
Prog	000143		000146	2						
Subr	000159		000159	1			2	2	2	
Subr	00016C		00017F	3			2		1	
Subr	000180		0001B2	8	3		2		1	
Subr	0001B5		00021F	6			2		1	
Subr	000233		000233	1			2		1	

Figure D.3 PROCAT Overview

```
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        0 0 0 0 0 0 1 1 1 1 1 2 2 2 3 3 3 3 3 4 4 4 4 4 4 4 6 6 6 7 7
        0 0 0 1 1 2 4 5 6 8 B 3 6 E 0 3 4 5 5 1 2 4 4 6 7 D 0 4 C 0 1
        0 3 B 3 B 3 3 9 C 0 5 3 0 D 3 4 5 2 F 4 1 0 C A 9 B 4 9 D 3 8
000000           I D D D D D D I I I I D I D I I I D I I D D D D
000003
00000B
000013
00001B
000023
000143
000159
00016C                             D
000180                           D
0001B5                                                 D D
000233
000260
0002ED
000303
000334
000345
000352
00035F                       D D D   D   D D D
000414
000421                             D       D
000440
00044C
00046A
000479       D                           D           D
0004DB
000604
000649
0006CD
000703
000718
```

lines   == calling routines
colums == called routines
D   == direct called
I   == indirect called

Figure D.4 Calling Matrix

In addition it furnishes an input file to the control flow analyzer COCAT. The overview table indicates possible deviations from required code attributes in number of exits, code sharing, use

31

of indirect calls, and recursion. The significance to verification depends on (a) the coding standards that are invoked, and (b) the weight given in the assessment to procedural vs. functional deviations. In the example code submitted by SoHaR a number of library routines (supplied with the compiler) used recursion. The standard to which the code was developed discourages recursion. The identification of this feature is considered desirable (as a caution notice), but it did not cause abandonment of the library routines. The calling matrix provides valuable insights into the calling structure and the types of calls that are made. Some compilers provide a similar matrix at the source code level that can be compared with that generated from the machine code. The hierarchy file is another useful tool for analyzing top level program flow, and can usually be compared with an equivalent hierarchy file generated from the source.

D.3   Control Flow Analyzer -- COCAT

COCAT analyzes the detailed control flow within program segments. Its main purpose is to identify structural weaknesses in the code, such as high levels of nesting, heavy dependence of backward jumps, unstructured jumps, and improper exits. Even when they are by themselves tolerable, the presence of deviations from good coding practice may indicate a need to revise the code. The presence of more than a few deviations in a program indicates at the very least that it may be very difficult to maintain.

The key outputs of COCAT are:

-    Control flow statistics (Figure D.5)

-    Comment sheet on exceptions from structured control flow (Figure D.6)

-    Graphic presentation of control flow (Figure D.7)

-    Pseudocode presentation of control flow (Figure D.8)

Examples from CoCAT protocoll (File *.PRC)

Number of Routine	Name	Kind	Startaddress	Number of Blocks	Nesting depth of forward jumps	Nesting depth of backward jumps	Nesting depth of structures	Number of jumps	Structure violation	No of weaknesses	Runtime [cycl.]	Paths
1	Main	p	0000	138	6	4	8	108	41	1	619	>999999
2	IR 0	i	0003	40	10	0	11	34	15	0	160	20
3	Timer 0	i	000B	21	5	0	6	17	2	2	124	13
4	IR 1	i	0013	13	5	0	6	11	2	0	65	6
5	Timer 1	i	001B	1	0	0	1	0	0	1	2	1
6	Serial Port	i	0023	1	0	0	1	0	0	1	2	1
7	zu Main	s	0800	5	0	1	2	2	0	0	72	4
8	zu 0800	s	0859	3	0	1	2	1	0	0	5	2
9	zu 0800	s	085E	3	0	1	2	1	0	0	5	2
10	zu Main	s	0870	8	2	1	4	5	1	0	38	7
11	zu Main	s	08B0	30	6	1	7	25	14	0	127	62
12	zu Main	s	08C0	30	6	1	7	24	14	0	125	62
13	zu 0870	s	09B0	3	0	1	2	1	0	0	13	2
14	zu 0870	s	09B2	2	0	1	2	1	0	0	12	2
15	zu 08B0	s	09D0	3	0	1	2	1	0	0	13	2
16	zu 08B0	s	09D2	2	0	1	2	1	0	0	12	2
17	zu Main	s	0A00	19	3	1	4	12	2	3	71	104
18		s	0A70	10	1	5	6	7	0	3	38	>999999
Total				332	10	5	11	251	91	1		

Overview of Controlflow Properties

0023 : interrupt routine consists of a RETURN only
024C : unconditional jump over 20 Byte to next block
0A5A : infinite loop consisting of 1 Block only
0A5A : routine ends with a jump instead of a RETURN
0A9A : routine does not begin at first block
0C09 : RETURN before end of Routine
0CA5 : unconditional jump to next statement

Hints on Controlflow Pecularities

Figure D.5 COCAT Statistics

Nr	Address + Name / Text / Comment	o.k. y	n
6	OA9A IR 0 routine does not begin at first block		
7	OC09 IR 0 RETURN before end of Routine		
8	OCA5 IR 0 unconditional jump to next statement		
Date:		Name :	

Comment Sheets for Hints on Controlflow Pecularities

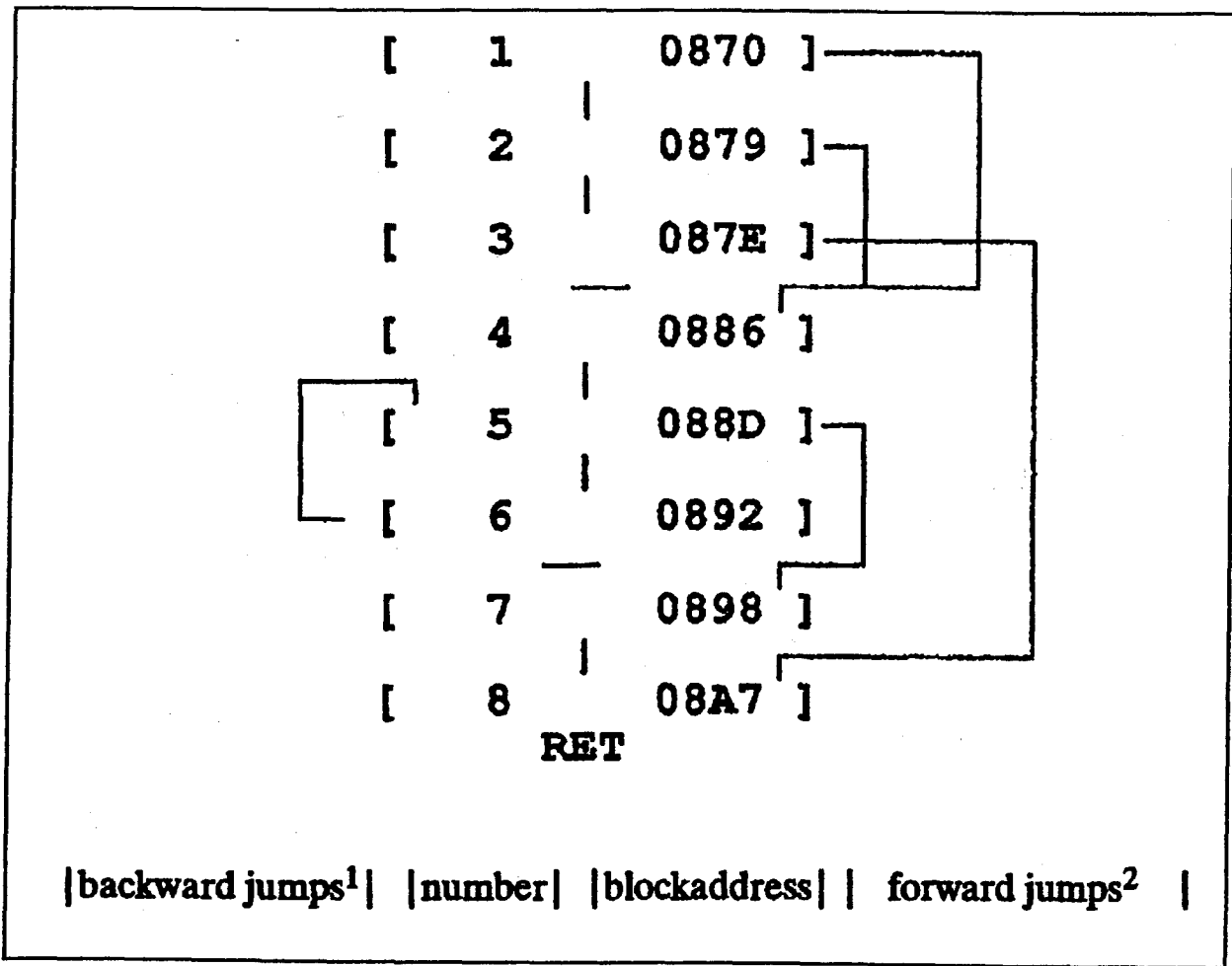Figure D.6 COCAT Explanations

Figure D.7 Graphic presentation of control flow

```
Example for Pseudocode Presentation of Controlflow


PROCEDURE  H_0870_  zu Main  ;
BEGIN
     Block 0870.a;
     IF 0870.b THEN BEGIN
     Block 0879.a;
     IF NOT 0879.b THEN BRANCH TO 0886  !!!;
          Block 087E;
     END
     ELSE BEGIN
  x?:     Block 0886;
          WHILE 088D DO BEGIN
             Block 0892;
          END;
          Block 0898;
     END;

     R E T U R N ;
END;


If controlflow is not structured 100% pseudocode will envolve jumps and the related labels.
Jumps are marked by "!!!" labels by "?". Thus they may be easy found with an editor.
```

Figure D.8 Pseudocode presentation of control flow

D.4   Data Flow Analyzer - DACAT

DACAT uses a file generated by DISCAT that identifies top level distributions of variables, and several files generated by COCAT that describe the control flow. From this it generates

-    a statistical overview of variables usage (Figure D.9)

-    a statistical overview of operand usage in a similar format

-    a read/write table for each variable (Figure D.10)

-    a pseudo code presentation of each routine with symbolic assignments (Figure D.11)

Examples from DaCAT protocoll (File *.PRD)

number of routine
name
kind
startaddress
assignments
conditions
calls + special instructions
different variables (total)
different read
different written
local variables
global variables
indirect var.
real-time
stack

#	name	kind	startaddress	assignments	conditions	calls + special instructions	different variables (total)	different read	different written	local variables	global variables	indirect var.	real-time	stack
1	Main	p	0000	278	77	7	79	43	61	18	61	0	0	6
2	IR 0	i	0003	59	18	0	44	26	26	7	37	0	0	8
3	Timer 0	i	000B	40	8	0	27	18	20	4	23	0	0	12
4	IR 1	i	0013	23	4	2	16	11	10	1	15	0	0	8
5	Timer 1	i	001B	0	0	0	0	0	0	0	0	0	0	0
6	Serial Port	i	0023	0	0	0	0	0	0	0	0	0	0	0
7	zu Main	s	0800	35	2	6	22	14	18	5	17	0	0	6
8	zu 0800	s	0859	2	1	0	1	1	1	0	1	0	0	0
9	zu 0800	s	085E	2	1	0	1	1	1	0	1	0	0	0
10	zu Main	s	0870	17	3	1	9	5	7	0	9	0	0	0
11	zu Main	s	08B0	48	15	2	22	16	19	0	22	0	0	0
12	zu Main	s	08C0	48	15	2	22	16	19	0	22	0	0	0
13	zu 0870	s	09B0	5	1	0	4	3	4	0	4	0	0	0
14	zu 0870	s	09B2	4	1	0	4	3	4	0	4	0	0	0
15	zu 08B0	s	09D0	5	1	0	4	3	4	0	4	0	0	0
16	zu 08B0	s	09D2	4	1	0	4	3	4	0	4	0	0	0
17	zu Main	s	0A00	37	9	0	15	13	13	6	9	0	0	0
18		s	0A70	25	6	0	11	6	11	3	8	0	0	0
Total				632	163	20	121	90	107	44	241	0	0	40

Overview of Dataflow Properties

Figure D.9 DACAT Statistics

Identifier	READ	WRIT	lo	glo	Mode
b00	4	4			2
b03		3			3
b09	1	2	X		
b0D	2	2			3
b0F	2	2			2
b11	2	2			2
b15		1	X		
b17	1	6			2
b19		1	X		
b1B		1	X		
b1F	1	4			4
b8E		7			2
b91		4			2
b93		2			2
b95	4				2
b97	2	14			7
bAF		2	X		
bD3		5			4
bE0	3				2
bE4	3				2
bE7	9	3			5
dptr	20	5			2
p[?]	20				2
r00	43	18			4
r02	8	3			2
r11	4	4			2
r14	5	2	X		
r16	3	2	X		
r1E	2	5			3

Identifier	READ	WRIT	lo	glo	Mode
b01	5	4			2
b08	1	2	X		
b0C	2	2			3
b0E	2	2			3
b10	2	2			2
b12	1	6			2
b16	9	9			2
b18		1	X		
b1A		1	X		
b1E	1	4			4
b8C		8			2
b90		4			2
b92		3			2
b94	2				2
b96		7		X	
bA9		7			2
bB1		3			3
bD4		4			4
bE1	1			X	
bE6	1			X	
cy	16	97			14
p[#B0]	1			X	
pc	3			X	
r01	7	6	X		
r03	12	6	X		
r12	4	4	X		
r15	5	2	X		
r17	3	2	X		
r1F	2	5			3

Figure D.10 Variable usage

Example for Pseudocode Presentation (File *.PAS)

```
PROCEDURE  H_0870_  zu Main  ;

BEGIN
     Block 0870.a
     b97:=#0;
     rE0:=r45;
     cy:=§CYSUB(rE0,#00);
     IF (r45=#00) THEN BEGIN
     Block 0879.a
     rE0:=r44;
     cy:=§CYSUB(rE0,#00);
     IF NOT (r44=#00) THEN BRANCH TO 0886  !!!;
          Block 087E
          r46:=rE0;
          r47:=rE0;
          r48:=rE0;
     END
     ELSE BEGIN
x?:     Block 0886
         rE0:=#04;
         CALL 09B2;
         r46:=#7E;
         WHILE (bE7=#0) DO BEGIN
             Block 0892
             CALL 09B0;
             r46:=r46-#1;
         END;
         Block 0898
         cy:=§CYRRC(r46,#0);
         r46:=§RRC(r46,#0);
         rE0:=r44;
         bE7:=§CYRRC(r46,#0);
         r47:=r44;
         r48:=r45;
     END;
        R E T U R N ;
END;
```

Figure D.11 Pseudocode Assignment Analysis

The statistical overviews can be used to determine adherence to provisions of the applicable coding standards, e. g., on the use of indirectly referenced variables. Another important use is to indicate the relative complexity of code segments, e. g., from the ratio of conditions to assignments, or the criticality of segments, e. g., by the number of global variable references. These inferences can guide the allocation of verification effort.

The read/write tables identify potential problem areas in the execution of the program, e. g., variables that are read but not written, or written but not read. Not all of these instances are programming errors, since the machine code does not distinguish between constants and variables, or between variables generated for use within the program and those generated for external use. But the identification provided by the table indicates where further analysis is required. The columns headed "lo" and "glo" in this figure denote local and global variables, respectively. The column headed "mode" permits designation of variables that can be accessed by bit-wise addressing.

The pseudo code is valuable as a starting point for further analysis required after examination of the previously discussed DACAT outputs. It can also be used as a backward reference to the source code.

D.5   Use of CATS for Analysis of a Real-Time Program

To aid in the understanding of CATS one module of a real-time program developed by SoHaR was submitted to TÜV Norddeutschland for analysis by CATS. The overall program involves multi-tasking and is intended to run on an 80286 or higher processor. The module submitted for analysis was separately compiled for an 80186 environment (and thus believed to be suitable for use with CATS) but the downgrading was not entirely successful and a number of 80286 instructions survived in the machine code that was analyzed. These instructions were recognized as anomalies and were the only outright deficiencies encountered in the analysis.

However, the CATS analysis also pointed to a number of weaknesses in the structure of the executable code that had gone unrecognized in the review of the source code:

-    re-entrant code in library routines

-    multiple entries to library routines

-    multiple exits in compiled code (due to the optimizing feature of the compiler)

-    use of nesting levels that were judged high by TÜV standards

After receipt of the results the implication of the weaknesses on the operation of the code were reviewed and it was decided that no changes were required. This does not detract from the value of identifying these potentially troublesome features. The most valuable function of the analysis

was the positive verification that the machine code corresponded essentially to the source code, and that it did not contain unspecified functions.

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.