

ORIGINAL

ACRST-1947

OFFICIAL TRANSCRIPT OF PROCEEDINGS

Agency:

Nuclear Regulatory Commission
Advisory Committee on Reactor Safeguards

Title:

Subcommittee on Computers in Nuclear
Power Plant Operations

Docket No.

LOCATION:

Bethesda, Maryland

DATE:

Tuesday, February 9, 1993

PAGES: 1 - 230

ACRS Office Copy - Retain
for the Life of the Committee

12-101

ANN RILEY & ASSOCIATES, LTD.

1612 K St. N.W., Suite 300

Washington, D.C. 20006

(202) 293-3950

ORIGINAL

ACRST-1947

OFFICIAL TRANSCRIPT OF PROCEEDINGS

Agency:

Nuclear Regulatory Commission
Advisory Committee on Reactor Safeguards

Title:

Subcommittee on Computers in Nuclear
Power Plant Operations

Docket No.

LOCATION:

Bethesda, Maryland

DATE:

Tuesday, February 9, 1993

PAGES: 1 - 230

ACRS Office Copy - Retain
for the Life of the Committee

12-101

ANN RILEY & ASSOCIATES, LTD.

1612 K St. N.W., Suite 300

Washington, D.C. 20006

(202) 293-3950

9302120262 930209
PDR ACRS
T-1947

PDR

PUBLIC NOTICE BY THE
UNITED STATES NUCLEAR REGULATORY COMMISSION
ADVISORY COMMITTEE ON REACTOR SAFEGUARDS

DATE: February 9, 1993

The contents of this transcript of the proceedings
of the United States Nuclear Regulatory Commission's
Advisory Committee on Reactor Safeguards, (date)
February 9, 1993, as Reported herein, are a record
of the discussions recorded at the meeting held on the above
date.

This transcript has not been reviewed, corrected
or edited, and it may contain inaccuracies.

ANN RILEY & ASSOCIATES, Ltd.

Court Reporters
1612 K. Street, N.W., Suite 300
Washington, D. C. 20006
(202) 293-3950

1 UNITED STATES OF AMERICA
2 NUCLEAR REGULATORY COMMISSION

3 * * *

4 ADVISORY COMMITTEE ON REACTOR SAFEGUARDS

5 * * *

6 Subcommittee on Computers in Nuclear
7 Power Plant Operations

8 Nuclear Regulatory Commission

9 7920 Norfolk Avenue

10 Bethesda, Maryland

11 Tuesday, February 9, 1993

12 The subcommittee met, pursuant to notice, at
13 8:30 a.m., H. Lewis, chairman, presiding.

14 Members Present:

15 H. Lewis, Chairman

16 J. Carroll

17 I. Catton

18 T. Kress

19 P. Shewmon

20 C. Wylie

21 Cognizant ACRS Staff Member:

22 D. Coe
23
24
25

P R O C E E D I N G S

MR. LEWIS: The meeting will now come to order. I have to read that. I'm not sure what coming to order really means in this context.

This is a meeting of the ACRS Subcommittee on Computers in Nuclear Power Plant Operations.

I'm Hal Lewis, subcommittee chairman.

There are other members of the Committee at the table: J. Carroll, Charlie Wylie, Ivan Catton, and Tom Kress; and there are consultants who I will ask to introduce themselves.

The purpose of the meeting is to hear from experts in the field of safety-related software development and quantitative assessment. This afternoon we will hear from some utilities about their experience in doing retrofits of their computer systems into I&C systems that were previously analog.

Doug Coe, to my right, is the cognizant ACRS staff member for this meeting. I believe he has the right to close the meeting if we get unruly, to throw us out, and to do whatever he wishes, because he actually works for the government.

This is a continuation of a series of meetings we've been having on the subject of digital systems in nuclear power plants. I don't have any particular

1 introduction to make except that there is a continuing
2 dilemma because digital systems are in the rest of the
3 technical community generally considered to be more
4 reliable, consistent, robust, and so forth, but not more
5 analyzable than analog systems.

6 The consequence is that the Nuclear Regulatory
7 Commission, which often errs on the side of believing that
8 better regulation equates to better safety, is concerned
9 about how to certify both digital systems for new power
10 plants and for replacements of analog systems with digital
11 systems in existing power plants. It's a dilemma we've all
12 been coping with and we have been holding a series of
13 meetings to try to address the issue. So the issues are
14 both the genuine safety and the appearance and certification
15 of safety by the regulatory agency for whom we are
16 functioning here.

17 Before we begin, I would ask our consultants to at
18 least say who they are so we all know who is sitting at the
19 table.

20 Dick.

21 MR. KEMMERER: I'm Dick Kemmerer. As opposed to
22 what is on the tentative schedule, I'm not from UCLA. I'm
23 from the University of California at Santa Barbara.

24 MR. LEWIS: You have to understand this is
25 Washington. They don't know where Santa Barbara is.

1 MR. KEMMERER: They started with it correct but
2 they changed it.

3 MR. CATTON: Some of us at the table do know.

4 MR. LEWIS: I have to tell you that as of the day
5 before yesterday Santa Barbara was under seven feet of
6 water. So it was kind of hard to find anyway. I apologize
7 for that. know where Santa Barbara is.

8 MR. CARROLL: Your reverse osmosis desalinization
9 plant produced too much water.

10 MR. LEWIS: Our desalination plant is in standby
11 and it's actually a very interesting example of decision-
12 making, because the decision to build it was correct at the
13 time and the fact that it turned out to be unnecessary in
14 retrospect doesn't change the fact that it was correct at
15 the time. This is a deep issue which we won't go into at
16 this meeting.

17 MR. PLACE: I'm Pat Place with Software
18 Engineering Institute.

19 MR. LEWIS: I think everyone has to learn to speak
20 into the microphone so people in the back can hear. More
21 important, so that the record will be intact.

22 MR. COBB: I'm Dick Cobb from Software Engineering
23 Technology.

24 MR. LEWIS: It says here you are going to be our
25 first speaker; is that correct?

1 MR. COBB: That's what I understand.

2 MR. LEWIS: Do any other members of the committee
3 have something to say before we begin? Consultants?

4 [No response.]

5 MR. LEWIS: Let's get on with the show. Rich Cobb
6 will start.

7 [Slides shown.]

8 MR. COBB: What I wanted to talk about this
9 morning was applying something that we call cleanroom
10 engineering, software engineering to the development and
11 certification of what we would consider correct software for
12 shutdown systems.

13 I'm Dick Cobb and I'm at Software Engineering
14 Technology. Harlan Mills who is the coauthor of this paper,
15 who is not here, until he retired from IBM was a fellow at
16 IBM and had probably made more significant innovations in
17 software development than anybody else, at least in my
18 opinion, in the United States.

19 The issues that I am going to try to address today
20 are summarized on this slide. As I understand it, these are
21 the issues that you are trying to address as a Committee.

22 First, is it feasible to produce correct software?
23 What we mean by correct software is software that will
24 provide its users with failure-free behavior where failure-
25 free means that the software performs exactly as its

1 specifications say it should perform?

2 The second issue is, if you want to produce
3 failure-free software, what engineering practices ought you
4 to follow in order to increase the probability of achieving
5 the goal?

6 Third, the bigger question, I think, is it
7 feasible to specify a correct function for the software,
8 what behavior you would like the software to perform?

9 Fourth, what engineering practices should
10 developers follow to increase the probability of getting a
11 correct function. "Correct" here is in quotation marks
12 because correct here is a judgmental issue about what the
13 behavior ought to be whereas the first issue is a
14 mathematical issue where it can be proven that it's correct
15 in relation to this specification that you decide here.

16 The fifth issue is, what's the probability of the
17 software satisfying the stringent reliability requirements
18 for nuclear power using these suggested practices.

19 Sixth is, what methods should the regulators
20 follow to determine the likelihood that the software to be
21 licensed is correct and will provide failure-free
22 performance as claimed by whoever the producer of the
23 software is, and what's the probability the regulators will
24 make a licensing error, i.e., that they will either license
25 a piece of software they shouldn't have licensed or that

1 they will not license a piece of software that they should
2 have licensed.

3 MR. LEWIS: As you go along, somewhere you will
4 deal with the size software package we are talking about. I
5 can produce a three-line program as perfect, but a million
6 lines I can't.

7 MR. COBB: I would like to wait. I think there is
8 one more slide before I try to deal with that issue.

9 MR. LEWIS: Fine. I was really setting an example
10 to prove to other people that it's legal harass the speaker
11 while he speaks.

12 MR. COBB: I like to be harassed, because that's
13 the best way for everybody to communicate.

14 The conclusion is we believe it's feasible to
15 produce and verify, prove mathematically that software, at
16 least of the sizes that we are talking about here, are
17 correct. I believe it's possible to produce correct
18 functions at least for the software components, and I will
19 explain what I mean by that in a few minutes.

20 A quantitative certification process exists that
21 can be used to assess the quality of the end product by
22 using statistical techniques like was applied to
23 manufacturing, which is new for software.

24 A process can be used to develop the functions and
25 the software can be reviewed and regulated by the NRC.

1 And complete verification proofs and certification
2 tests can be reviewed and regulated by the NRC, which gives
3 the NRC complete visibility to what is going on. In fact,
4 we would claim that software can be the most reliable part
5 of a system like a shutdown system.

6 I think that these claims are not based on wishful
7 thinking but based on the application of sound engineering
8 practices, which is new for people in the software
9 profession. It's rooted in a firm mathematical foundation.
10 The principal inventor of these techniques was Harlan Mills.
11 He worked for IBM. IBM had cleanrooms where electrical
12 engineers did fine-tuning assembly. They kept all of the
13 contaminants outside and didn't let them get into the
14 electronic assembly. His idea on software development was
15 don't put the failure in in the first place; keep it out so
16 you don't have to debug it out.

17 What we mean by correct software I've already
18 said. It provides exactly the behavior defined by a
19 specification, and if it's correct, it will provide failure-
20 free performance.

21 People cannot look at a piece of software and say
22 is this piece of software correct. We might make proofs.
23 We can do all of those sort of things. We can look at the
24 proofs and we can reason about the proofs, but if software
25 is indeed correct, it will provide failure-free performance.

1 People can observe failure-free performance.

2 So the software development process is the process
3 of constructing a rule for mathematical function such that
4 as the software is executed it provides the specified
5 behavior. A piece of software is nothing more and nothing
6 less than a mathematical writing. It's a rule for a
7 function. The specification must define the function. To
8 have correct software you have to show that the behavior
9 that is executed by the piece of software provides exactly
10 the same behavior as the function defines.

11 We all know that a function is a set of ordered
12 pairs where the first member of the pair is an argument and
13 the second number is a value. A rule for a function is a
14 way of giving an argument to be able to calculate a value.
15 That's exactly what we do with a piece of software. We give
16 it a stimulus and we get a response; we give it a stimulus
17 and we get a response.

18 If we can define a mathematical function for what
19 the shutdown system is supposed to do, then we can develop a
20 rule and verify and prove that the rule is correct for that
21 function. As well know, there is essentially an infinite
22 number of correct rules for any function. As people who
23 develop software know, there are lots of also nearly correct
24 rules too, and it seems like there are more of those than
25 there are correct rules.

1 The craft-based practices that we use today.
2 We've only been developing software for about a human
3 generation, about 40 years. At the end of 40 years civil
4 engineers hadn't invented the right triangle. We've done
5 pretty good in software, but the practices we have today are
6 not capable of doing the job; they result in error prone
7 code.

8 Engineering-based methods for software development
9 based on a science foundation are now available, and I'll
10 talk about the history of those developments. Such methods
11 can be used to develop correct rules expressed in a chosen
12 programming language for a given function. I've already
13 said the bottom point.

14 As Dr. Lewis observed, the simpler the required
15 behavior the easier it is to develop and verify that the
16 software is correct.

17 As I understand it, the designers of shutdown
18 systems only require software components to perform a single
19 clear-cut function. It will be easier to develop and verify
20 correctness of the shutdown software than it is for most
21 other projects where the software is being asked to perform
22 a multitude of complex functions.

23 In most project situations the emphasis is on
24 developing a correct program, not in developing a proof for
25 correctness. I think in this situation the stress is on

1 developing both a correct program and recording a proof of
2 correctness so all interested parties can evaluate the
3 proof. Just like Euclid wrote down some theorems in
4 geometry and he set down the proofs so everybody could
5 understand them and everybody could reason about them and he
6 put them out for public literature, in our kind of
7 terminology a program is supposed to be for public
8 literature. It's not for private consumption.

9 The effort of documenting a complete proof of
10 correctness is completely proportional to the size. Since
11 the designers are planning small, simple, straightforward
12 programs, I think that is well within people's capabilities.

13 For instance, my understanding from talking to
14 some people who are doing things with shutdown systems is
15 that we are talking about a typical shutdown system not in
16 full detail. You are going to get down here to some voting
17 to decide whether to trip the reactor. This voting is going
18 to include some analysis of average power, some analysis of
19 reactor limits, some analysis of area radiation.

20 To get to that analysis function which is quite
21 straightforward and simple you've got a lot of sensors out
22 here someplace and the data comes back by some multiplier.
23 So you've got a software component here a software
24 component here, a software component here, a software
25 component here, and then you've got this emergency core

1 cooling, or whatever that is, and you are going to have some
2 voting in order to decide to send these controls mechanisms,
3 and you have got to send stimuli to other channels.

4 In this system there are at least 14 separable
5 simple software functions that have to be developed, each
6 one defining a simple well defined thing. The estimate is
7 that maybe this involves a total of 12,000 lines of code.
8 That may be high. Some people have said as high as 20,000.

9 Each software item is a small single well defined
10 function. There is no requirement for any failure-prone
11 operating systems like Unix or MS-DOS; there is no
12 requirement for any quat software like a database management
13 system; it's just a simple little embedded piece of software
14 that -- I don't want to make it sound so simple, because
15 there are some challenges. I've seen some of the people who
16 have testified before you before. They talk about these big
17 projects. We're not talking about a big project. We're
18 talking about small self-contained projects, which makes it
19 simple to record the proof of correctness.

20 There are many examples of such small well
21 contained software. Even big monolithic systems like
22 database management systems that are failure prone have
23 little parts, the parts that the designers really cared
24 about, that they reasoned carefully about, that have never
25 exhibited a failure.

1 This is within our limits of capability to do.

2 I think that tries to answer your question about
3 size and complexity.

4 MR. LEWIS: Except that in the end one has to be
5 quantitative about these things. It's easy to define the
6 extremes of a thing so simple that we really can spell them
7 out and things so complicated that there is no hope, but in
8 the end we are going to be dealing with real systems, so we
9 will have to come down to being quantitative about the
10 subject.

11 For example, there are some steps left out here.
12 There is no compiler that doesn't have bugs in it somewhere.
13 The question is where you find them. At home the night
14 before last I called up the IBM bulletin board system to
15 find out how many bugs there were in the OS/2 operating
16 system, and it turned out I could download, I think it was,
17 12 one-megabyte disks full of bug fixes which I could apply
18 to my system. That's an awful lot of bugs. By our
19 standards that's really not that big a system.

20 MR. COBB: I don't mean to be argumentative, but
21 OS/2 is a pretty big system and it has been put together by
22 a bunch of hackers working all over the place and it is not
23 one of the better examples, in my opinion, of IBM's effort.
24 A better example might be the IBM Wheelwriter typewriter
25 which has 79,000 lines of microcode in it. It was released

1 in 1980. It was developed using some of the techniques that
2 we are talking about today. There has yet to have been a
3 report of a failure in that particular software. That
4 software was developed by a small integrated team. The
5 software was completely proven and it was loaded up.

6 Another operating system that I just participated
7 in a functional review on last week is being developed.
8 It's just going into testing now. In the first three weeks
9 of testing with an enormous number of test cases it has yet
10 to experience a failure in the software of some 300,000
11 lines of code. This is much more complex system in some
12 ways than OS/2.

13 MR. LEWIS: The statement that something has yet
14 to experience a failure isn't quite a mathematical proof
15 that it's a correct system.

16 MR. COBB: That's true. Let me talk a little bit
17 more about the proofs in just a few minutes if I might. I
18 would estimate the reasoning about the proofs for a
19 shutdown system like you are talking about would be about
20 the same number of proofs that would be in a one semester
21 mathematics course in a college course. That's about the
22 level of effort that would be required to do this work, to
23 put that into perspective. There are lots of textbooks.
24 You've got to look at the field of mathematics. You can't
25 look at the field of computer programming when you are

1 reasoning about these things

2 It's our opinion that in 1993 we are capable of
3 engineering failure-free software.

4 I told you that Harlan called these ideas
5 cleanroom software engineering. The way we develop software
6 is we start with our function. That's sort of unique in
7 software development. You start with a function and then in
8 a series of small refinements we follow each refinement by a
9 verification and the results are to try to keep design
10 faults out.

11 Stepwise refinement and verification algorithms,
12 as you all know, are a common way of going about solving
13 problems. Probably the most common way we think about
14 stepwise refinement and verification algorithms is long
15 division. In long division we start and we make an
16 invention and then we follow it by a verification. If it's
17 correct, we proceed; if it's incorrect, we adjust our
18 invention and we proceeding digit by digit until we decide
19 we're done. Third and fourth graders get a long fraction of
20 long division problems correct. Great minds like Archimedes
21 could not do long division at all.

22 The simple thing is it's process of the process,
23 the algorithm. What Harlan has managed to do is develop a
24 stepwise refinement and verification algorithm that works
25 for software systems of any complexity.

1 Just to try to compare craft-based development and
2 engineering-based development, those of you who are familiar
3 with software development will understand this.

4 In craft-based development a program is regarded
5 as a set of lines of instructions. In engineering-based
6 development a program ought to be regarded and is regarded
7 as a correct rule for a function. Today programs are
8 normally recorded using structured concepts. As you will
9 see in a minute, that's been an enormous improvement in
10 quality, just the introduction of structured programming
11 constructs. Over here they are always recorded using
12 structured constructs. Today at least all programmers feel
13 guilty when they don't write a well structured program.
14 They don't always do it, but they always feel guilty.

15 A specification is regarded as a list of
16 requirements that leave many details for later invention,
17 and specifications are really often considered as a burden.
18 People start doing development before they have a complete
19 specification.

20 In engineering a specification has to be regarded
21 as a clear description of the desired behavior, which means
22 that you need the mathematical function.

23 It turns out that a black function is the best
24 form of a function to use to describe behavior for a
25 software system. For any system, for that matter. No

1 details are omitted. Specifications are expanded to code
2 when they are complete. The whole intellectual process
3 moves from doing the coding to doing the specification,
4 which is the crucial issue.

5 With current programming practices there is no
6 formal transformation processes known or used. Debugging is
7 normally used to verify code. Debugging is the most error-
8 prone activity we do when we are doing software development.
9 It's well known in the very best software development shops
10 today that between one time in ten and one time in 20
11 whenever you make a fix to a piece of software you introduce
12 another fault that can result in a failure, that is harder
13 and deeper than the correction that you made. Say 15
14 percent. That's a terribly hazardous operation to engage
15 in.

16 The one thing that you would like to be sure if
17 somebody is developing high quality software for you is that
18 they make as few changes, they do as little what we call
19 debugging, as little unit testing as they possibly can,
20 because that is something that should be avoided at all
21 possible cost.

22 When you are engineering software the specified
23 function is transformed into code in a series of small
24 refinements. Each refinement is followed by a proof. The
25 whole goal is to develop a correct rule. That's in the top

1 of your mind all the time.

2 Today failures are basically expected and
3 accepted. We start coding earlier because we have a lot of
4 debugging to do. When you are doing engineering you want to
5 have the attitude that failures are not acceptable; they
6 shouldn't get in there in the first place, and as soon as
7 they get in there you want to find them and get them out as
8 soon as you possibly can, in the next verification.

9 Programs today are largely regarded as private
10 art. What we want to do is make programs public literature.

11 The way we work and the way we recommend you work
12 is as a team. Somebody makes a small refinement. You've
13 got a small engineering team, three or four people. The
14 person who is making the refinement and then the proof of
15 verification presents it to the team. As we all know, a
16 mathematical proof is an experiment. The major subject of
17 the person doing the proof is the person doing the proof
18 himself or herself in order to make sure that they are
19 correct. This leads to simple ways of doing it, because if
20 you are going to have to prove something, you want to get a
21 simple way of doing it.

22 We follow every expansion a proof. In your
23 particular case I think all of those proofs should be
24 formally recorded and defined for posterity so that if
25 anything ever goes wrong in any part of the software you'll

1 know what happened.

2 In craft-based development testing is basically
3 used as a means to find program faults.

4 Over here we have a hypothesis that the
5 development team has been infallible and has developed
6 correct code. We have a hypothesis of correctness, so we
7 test it for that way.

8 In today's testing environment tests are designed
9 to cover all program paths. This is a very difficult thing
10 to do. Even if you have a simple square root program on a
11 64 bit processor and you would like to cover all
12 combinations of stimuli history of that program, you have to
13 run nearly two to the 20th test cases. That is a tremendous
14 number of tests. You've got to have a scientific basis for
15 doing that. So anecdotal arguments are used to extend
16 observed results to the population.

17 Over here software is evaluated using a random
18 sample of expected usage. A real proper random sample just
19 like we are doing quality control in a manufacturing
20 proposition. It turns out that a simple stochastic process,
21 a Markov model, can be used to define the probability
22 density function for use of a software system. You can
23 constructive generate the entire probability density
24 function. So sample reliabilities can be used to estimate
25 population reliabilities.

1 The statistical problem we are faced with is let's
2 assume we have a software system; it has run 100,000
3 failure-free uses; never has failed in its entire history.
4 How do you project that? What is your feeling about the
5 probability of that system ever failing in the future? That
6 now is the most intellectual challenging problem we have, to
7 solve that problem.

8 MR. LEWIS: If you have an input function n bits,
9 there are 2 to the n th possible inputs; therefore there are
10 2 to the n th possible functions that can be implemented.
11 These are trivial numbers. In order to test an n bit input
12 you have 2 to the n th possible inputs which you want to
13 study. You propose to do it through a Markov model, which
14 means a random sampling model on the inputs, as an
15 alternative to a complete study of the input space. That
16 would then populate the output space rather thinly.

17 The probability density you are talking about is
18 the probability of having hit the place where there is a
19 problem or finding a problem if there is a problem. This is
20 certifying the correctness with a certain probability level,
21 and I worry about the application of this to a regulatory
22 environment, because I can tell you the NRC staff is not
23 going to settle for saying there are four 9's probability
24 that the reactor won't melt because of a software error.
25 That's the kind of problem we are going to have to address

1 as we go along.

2 I'm just trying to keep this quantitative.

3 MR. COBB: I understand. That's a profound
4 scientific question that you have just asked. I now you
5 realize that. The question is, for any physical system --
6 forget whether it's software or not -- I have a design.
7 Today what we do in hardware systems is basically reason
8 about the correctness of hardware systems analytically
9 before we do testing. The testing that we do is to confirm
10 the existence of the hypothesis that the design was in fact
11 correct.

12 In a software system, not only do you have to test
13 the entire input space, but you have to test all the
14 combinations of prior history, all the ways that that input
15 can occur. A program exists in real time even if it's
16 performing a batch function. The only thing a program knows
17 is all the stimulus it has ever received since it has been
18 invoked. That's the only thing it knows. That's its only
19 intelligence, including initial conditions, and you've got
20 to think of them as a stimuli.

21 Not only do you have to test each stimuli; you
22 have to test every combination of stimuli that it could have
23 received. It's not just one stimuli; it's the whole
24 combination, in every different combination, because this
25 program has memory. You have to reason about those

1 combinations. So this concept of coverage testing just
2 doesn't work in software. It's not that kind of object.

3 The whole advantage, as you know, of computer
4 software is it does have memory and it's the memory that can
5 get you into trouble, and it's those combinations of
6 stimuli. Just like if you are worried about a reactor
7 vessel, the strength of materials failing. That vessel has
8 memory too. It wears out in little pin holes; it just
9 doesn't wear out all uniformly. It's exactly the same
10 problem. I worried about that problem not in reactor
11 vessels but for vessels that contain catalytic crackers.
12 It's a very difficult problem.

13 No matter how you would like to make it a non-
14 statistical problem, it is a statistical problem, and
15 statistics is a very powerful branch of mathematics.

16 MR. LEWIS: I never use reactor vessels as an
17 example when Paul Shewmon is sitting at the table. He knows
18 too much.

19 MR. COBB: I'm sorry. I know a little bit about
20 software.

21 MR. PLACE: Before you continue, your discussion
22 of using a Markov model for determining probability of --

23 MR. COBB: To define the probability density
24 function of use.

25 MR. PLACE: That seems to be treating software

1 purely as black box. If you look into the contents of the
2 software, surely you can construct a set of tests that are
3 more likely to give you some confidence in a hypothesis of
4 correctness than by using a random sample. We have boundary
5 conditions that you need to test for much more than you can
6 test for the standard case. Let's take division. In my
7 division case, if I assume my software is developed
8 correctly, I would like to check division by zero to
9 division by a negative number and division by positive
10 numbers rather than choosing with the Markov model from the
11 sample space of all possible dividends.

12 MR. COBB: I agree with that. What that means is
13 if there are particular spaces, what you really want to do
14 is test different strata of possible use. One of the strata
15 in a division case that you would like to test is strata
16 that are near the boundaries, because that is where you
17 would be. Just like you do sampling of a population to
18 assume some thing, you want to sample where the boundaries
19 are. You also want to sample not necessarily near the
20 boundaries when you are sampling near the boundaries,
21 because you want to get that history of sequences. If you
22 approach the boundary from this place or that place, does it
23 make a difference?

24 Yes, you want to test where you think the failures
25 are going to be most profound and things like that, but

1 you've got to do that by defining the right strata of use.

2 I think that is an important point and it's a
3 point that I wasn't going to make in this 80 minutes because
4 I had to choose to make some points and not make others.
5 But that's exactly what you have to do.

6 When you are testing software and you are proving
7 correctness you do want to regard the software as a black
8 box. That's absolutely right. Because the only thing that
9 we can observe is correct is behavior.

10 MR. SHEWMON: Before you stop, let me introduce
11 something else. I'm the person that's supposed to know
12 something about pressure vessels but nothing about software.

13 One of the things that has gotten computers in
14 trouble is when somebody shorts out the power supply or
15 there is a low voltage; there is something referred to as a
16 light bulb incident where somebody was changing a light bulb
17 and shorted out a power supply and that just drove things
18 all kinds of funny ways.

19 Would you say that that is not your bailiwick?
20 What do we do to take account of that? Is that supposed to
21 be then given to you as part of your input on what the
22 possible signals are even though those signals might in this
23 case lie out of what would come from normal communications?

24 MR. COBB: As we all know, one of the important
25 parts about a function is its domain. I think there are two

1 questions that you have asked. The function must define
2 what to do with all stimuli within the expected domain and
3 what to do when stimuli are outside the domain and the
4 function has to define what to do with these things. In
5 this particular you clearly would want to prove complete
6 correctness, which means that you have to worry about the
7 domain boundary very carefully.

8 As far as the stimuli going in and out of bounds,
9 that's something that can be handled.

10 The other question that I wasn't sure whether you
11 were asking is, what happens when I have a power failure and
12 the computer completely turns off and I no longer have a
13 computer to do anything with? Somehow or other the whole
14 system's design has to take that issue into account.

15 MR. SHEWMON: There is an awful lot of concern
16 about what is called station blackout in power plants and
17 there are belts, suspenders and several other things to make
18 sure that there is guaranteed power supply. I think one
19 could show that that has a lower probability of occurring,
20 or I would at least say that it does, than the out of spec
21 voltage that gets introduced inadvertently.

22 MR. COBB: I don't know what the specifications
23 say to do, but let's assume that the specifications say that
24 if there are four channels that are voting and we have two
25 out of spec voltages and we still want to shut down, then

1 that is what we should do, shut down. If we say that we
2 want to have three out of spec voltages and shut down, then
3 we shut down. If we say we don't want to shut down for any
4 out of spec voltages and keep on running, then that's what
5 the software should do. But that's a specification issue of
6 the software.

7 I'm not claiming the specification is easy, but
8 that is where the focusing should be, getting the right
9 specifications. Let's assume that you are making the
10 specification about what to do about these out of spec
11 voltages and you're right, because you are always right
12 about such things, then it's up to us to implement precisely
13 those specifications and know that we have implemented them.
14 But make those specifications abundantly clear. In software
15 today, when we don't make our specifications absolutely
16 clear with a black box function people get confused about
17 what the software really does do because the specifications
18 aren't clear. That is a common failure today in software.

19 MR. SHEWMON: Thank you.

20 MR. COBB: Projects today are basically organized
21 around individual crafts people working on performing
22 activities. Here we want to organize projects around
23 engineering teams. We talked about teams producing
24 documents. Our whole goal is to perform this activity under
25 the intellectual control of the project participants.

1 That's an important ingredient. I think that is one of the
2 things that characterizes engineering from crafts people.

3 Sort of the history of how this has all been done
4 and where we are today is that in 1969 when Dykstra
5 introduced the concept of using structured programming
6 constructs for designing rules of functions he was
7 absolutely ridiculed. Nobody could write programs using
8 structured constructs. It couldn't be done and it wasn't
9 necessary. Today we know that that's right.

10 In the early 1970s we talked about stepwise
11 refinement, top down design and functional verification.

12 By 1980 these first three things were well
13 ingrained in software development. I will show you in a
14 minute what the impact is.

15 The biggest problem with all of this revolution,
16 which was a major revolution in program development, was
17 that we didn't know how to define the function. We didn't
18 understand what the form of the function ought to be.

19 Beginning in the 1980s Mills and others began to
20 think more about what is now called the mathematics of box
21 structures, and it turns out that there are only three forms
22 of functions that you need to worry about when you are
23 worrying about software: a black box; a state box, which is
24 a generalization of a state machine; and a clear box, which
25 is a process.

1 In another paper that I have passed out some of
2 those things have been defined for you in more detail.

3 They began to be reduced to engineering practice
4 in the late half of the 1980s in many places.

5 Then in 1989 Harlan published a paper called
6 Designing Systems with Box Structure Mathematics, or
7 something, which is a stepwise refinement and verification
8 algorithm that lets you reason from a black box function to
9 code in a series of stepwise refinements with verifications.
10 All of the mathematics to be able to do verifications in box
11 structure systems are there. That design algorithm was
12 published and now it's in the process of being introduced to
13 engineering practice. That paper was in June 1989 in IEEE
14 Computer.

15 The certification issue of using usage testing.
16 People began to really begin to focus on usage testing as
17 opposed to coverage testing in the early half of the 1980s,
18 which began to be introduced by engineering practices in
19 such places as Bell Labs, IBM and others.

20 The big problem at that time was we knew that we
21 wanted to do usage testing, but how do we define the
22 probability density function? It was in 1990-91 that the
23 theory of being able to use Markov models was developed as
24 part of a DARPA program using Markov chains for that. It's
25 now being introduced in engineering practices. In hardware

1 manufacturing, like IBM, telecommunications people, some DoD
2 sites it is really beginning to take hold. It's a very
3 important idea.

4 Specifications. The idea was to get a black box
5 function as part of your specifications began to take hold
6 in the second half of the 1980s. Then a six volume
7 specification, which I will talk about in a minute, was
8 introduced in the early 1990s.

9 Software development as a process. In the middle
10 1970s Harlan was responsible for defining the process that
11 is being used down in Houston to develop the space shuttle
12 software, and that is supposed to be a very fine process,
13 developing failure-free software. They've had some
14 failures, but and large it has been pretty successful. We
15 had one in-flight failure which was a result of a fix to a
16 fix to a fix. But that system is frozen in time for what
17 Harlan knew in the 1975 time frame.

18 I talked about this Wheelwriter failure-free
19 development. That was done using structured programming,
20 stepwise refinement, top down design and functional
21 verification. Complete proofs. There are lots of instances
22 of software being developed like that in IBM and other
23 places, but the only person to win two gold medals for
24 government service has used those same four practices to
25 write code for part of the government. His name is Paul

1 Friday who works in the Census Bureau.

2 Then cleanroom was introduced as ~~Markov~~ by
3 Mills and Linger in the second half of the 1990s. ~~That is~~
4 now a completely defined process for that which really
5 helps.

6 So there has been a real convergence in
7 development of a mathematical theory. That doesn't mean
8 that there hasn't been missteps and sidesteps doing that,
9 but we have moved forward.

10 MR. LEWIS: I apologize for being the slowest
11 person in the room. Could you distinguish for me between a
12 Markov chain and random testing of the space? What is
13 Markov about this?

14 MR. COBB: If you are going to do random testing
15 you must know what the probability density function is. You
16 want to get a random sample that is representative. In
17 order to select a random sample in the laboratory you need
18 to know what the probability density function is of use.
19 How do you define that density function? It's not just a
20 normal distribution; it's whatever it is. A Markov model
21 can be used to define that density function constructively.

22 MR. LEWIS: I'm just wondering what is Markov
23 about it. I understand that you want to not test the input
24 space just with a uniform distribution; you want to test it
25 according to the probability of use. That makes sense. But

1 what is Markov about such a prior density?

2 MR. COBB: In order to develop a Markov model in
3 order to be able to do this we have to be able to define the
4 states that the system can be in from the standpoint of an
5 outside observer. Those become the program states. We can
6 observe those as an outside observer. If we are in state I,
7 then we want to know what the probability is if we are in
8 state I that we will move to state J or K or L.

9 MR. LEWIS: So you are going to use a Markov model
10 to not just test the probability density of the input state,
11 but also the correlation between one input state and the
12 next input state?

13 MR. COBB: That's right.

14 MR. LEWIS: I see.

15 MR. COBB: A Markov model can have memory.

16 MR. LEWIS: I understand.

17 MR. CATTON: So it's a chain through the system.

18 MR. COBB: Sure.

19 MR. LEWIS: I understand what a Markov model is,
20 but I didn't realize you were then going to two dimensions,
21 because then you are not only studying the input space
22 density in your probabilistic testing, but you are also then
23 using the Markov model which you get from somewhere to
24 determine what the next state is that you study.

25 MR. COBB: Yes.

1 MR. LEWIS: Okay.

2 MR. COBB: In a test case that we would recommend
3 that you run is a complete usage. A usage state might go on
4 for hours as you keep generating test cases where you move
5 from state to state. One of the questions that has to be
6 answered about this particular issue is what is a use. Is a
7 use one set of failure conditions, or is a use some other
8 combination? There are some interesting engineering
9 questions.

10 MR. LEWIS: I'm just trying to understand this
11 quantitatively, because, again, with an n bit input there
12 are 2 to the n th possible input states. If you also
13 introduce correlations between one input state and its
14 following input state, which you have to do for the Markov
15 model, then we are squaring the number of options for which
16 we have to have a probability distribution, because we need
17 the Markov matrix as well as the initial probability density
18 on the input space.

19 MR. COBB: Right.

20 MR. LEWIS: I don't know whether by Markov you are
21 thinking of only two states or whether you are going to have
22 a multistate Markov.

23 MR. COBB: No. The relational database system IBM
24 sells, dB2, needs about 3,000 states, which is a management
25 size model.

1 MR. LEWIS: I'm beginning to founder on the
2 technical difficulties of implementing this, but please go
3 on.

4 MR. COBB: If you start doing this, I think you
5 will find out that it scales up; you'll find out that it
6 works and it's a decent theory.

7 MR. KEMMERER: Richard, are you going from state
8 to state or from component to component?

9 MR. CATTON: That's what I missed too.

10 MR. KEMMERER: In Harlan's paper that was in
11 Computer this month he talks about going from component to
12 component as opposed to state to state, which is going to be
13 a larger space, larger matrix.

14 MR. COBB: What is the definition of component?

15 MR. KEMMERER: I'm going from Harlan's paper which
16 I read on the way back here. So it's new to me. Component
17 is a software component of which you take these components
18 to make your system.

19 MR. COBB: We have to do two things here, I think.
20 First of all, I think you ought to use this kind of usage
21 testing for two different attributes of this system. One,
22 we want to test the software that is in this analysis
23 system. There the states are that the states can be in that
24 this analysis system can be in based upon all the inputs it
25 has received here and issuing the responses over here. So

1 we first want to verify that this analysis function performs
2 the function that we want it to do.

3 Then we also want to look at what is the behavior
4 of this system from here to here. Now the states we want to
5 look at are the states that this entire system can be in of
6 all of these components.

7 Does that answer your question?

8 MR. KEMMERER: No. In looking at Harlan's paper,
9 he's talking about software components and then he has a
10 probability that we are going to go from component A to
11 component B.

12 MR. COBB: Right.

13 MR. CATTON: Excuse me. Would that mean going
14 from MUX to the next MUX?

15 MR. COBB: That's what that would mean here.

16 MR. KEMMERER: We've got a straight flow here.
17 That's going to be one on those. That's really
18 uninteresting. The interesting thing is if I'm going from
19 one component to another component, for instance, if I have
20 straight line code or if I have a decision, that space is
21 going to be a lot smaller than the Markov matrix that I'm
22 going to have if I start going from state to state, because
23 I don't just say if I'm at this component what's the
24 probability I'm going to go here. If I'm at this component
25 with this history --

1 MR. COBB: That's what you have to do, sir.

2 MR. KEMMERER: That's what I thought you were
3 saying now, which is different than what I read in his
4 current paper.

5 MR. CATTON: Could you just summarize what you
6 just understood?

7 MF. KEMMERER: Let's take a simple component that
8 I'm deciding what to branch to next, what piece of software
9 to branch to next based on an IF statement. If I'm in a
10 state where one of the things I know about the state is that
11 a condition is going to be true, the probability that I'm
12 going to the next component is always 1 as opposed to if I'm
13 sitting there without knowing anything about the state. I
14 have separate probabilities to go to each component. I
15 think knowing which state is going to be a bigger matrix,
16 state to state versus component to component, because not
17 only am I in this component and how many different states
18 can I have, or as Richard was saying before, how many
19 different histories can I have. So this is a much bigger
20 set that you are starting off with here.

21 MR. COBB: The Markov model as you go from state
22 to state really generates memory about how you've been there
23 from before. Let's think of the Markov model as a square
24 matrix. It's always a square matrix. The states that you
25 have to define in this model are the states that the

1 software can be in or the system can be from the vantage
2 point of an outside observer.

3 Fortunately, in this particular system these
4 little components are going to have very few states. The
5 good thing about the people who think about these things is
6 that they have thought of a fairly simple system. They
7 haven't got a real complex system. Which is good. I guess
8 they did that because it was a good idea. I'm sure that's
9 the reason.

10 Sort of a small history of what's happened with
11 these different ideas. Back when people had been following
12 the traditional way of doing programming when we first
13 started, the defects after the software went under
14 configuration control that were measured might be between 50
15 and 60. When we went to structured programming, a top down
16 design with code inspections instead of functional
17 verification we may have had 20 to 40. When we substituted
18 functional verification for code inspection, we went to less
19 than 5 development defects per KLOC. People now are
20 beginning to talk about getting down to less than one here,
21 but the date of these sort of 1993 things are sparse still.

22 Operational defects, we may have 15 to 18. Modern
23 methods, we may have had 2 to 4. Our early cleanroom,
24 certainly less than 1. Now these are getting way less than
25 1.

1 Productivity. To the surprise of many people,
2 when you do a more formal job of doing your development and
3 seemingly doing more work productivity goes up. The reason
4 is that you are not doing rework. So not only do you get
5 higher quality, you get higher quality for less money of
6 investment. That's going to necessarily be the case in your
7 particular case, because you really want to formally record
8 all of these proofs and drive this to absolute zero with
9 certainty. You may not get 750 lines a month from people
10 who do that. But the software is fairly simple.

11 The message is that engineering software really
12 does work; it does produce higher quality; and it does do it
13 with increased productivity, which is a good message for
14 management. A good message for everybody.

15 One of the things that I think that says is that
16 not only can a manufacturer of one of these plants use these
17 ideas for the shutdown system, but they might use it for
18 their whole control system, and then the whole plant might
19 be much better off as a result.

20 I want to separate this issue of where failures
21 come from. There seems to be confusion about sources of
22 failures. There is no question a failure occurs when the
23 system does not provide the required service. If we talk
24 about this whole shutdown system, a failure occurs if it
25 didn't work when it is supposed to.

1 A specification failure occurs when the system
2 does not provide the required service because the system
3 designer did not account for some circumstance or otherwise
4 specified an improper or incomplete function to describe the
5 desired behavior.

6 A software failure occurs when the system does not
7 provide the required service because the software designer
8 introduced an unintended deviation from the specified
9 behavior.

10 In our situation, a software failure comes about
11 because of this reason; a system specification failure comes
12 about because of this. I think it's important to separate
13 these two sources of failures for this system.

14 A lot of software system failures also occur when
15 the software designer and the system designer do not fully
16 communicate because the specification is imprecise,
17 incomplete or otherwise misleading. That's one of the
18 things that you want to get away from. You want to get away
19 from these imprecise functional list of requirements and
20 things that we know are common in software projects. You
21 want to have a clear function, a mathematical function that
22 is rigorously defined.

23 A lot of people talk about that. All the people
24 who are leading thinkers in this area now are stressing the
25 importance of doing that.

1 MR. LEWIS: We are guilty of one specification for
2 which we have not provided adequate implementation. I
3 figure you are about a third through your viewgraphs and
4 two-thirds of the time we allotted. I don't know whether
5 the fault is in the specification or in the software, but
6 there is clearly a problem. I only call it to your
7 attention.

8 MR. COBB: Thank you. I was well aware of that.
9 We've have had some questions which have been good.

10 MR. LEWIS: It's also our fault.

11 MR. COBB: I'm not trying to say it's your fault.
12 I think some of the later slides will go faster.

13 It's very difficult with today's process to say
14 how many of the system failures we have are a result of
15 software and how many are a result of specification
16 failures. What we do know is what I've already said. When
17 you make a change to the software that's the largest source
18 of software potential failures. So you don't want to do
19 that.

20 One of the recommendations is that we clearly have
21 to separate the specification problem from the
22 implementation problem. Specifications have to define the
23 desired behavior and software has to define the rule for
24 that behavior. In fact, one of the things I am going to
25 recommend in a minute is that the Nuclear Regulatory

1 Commission ought to license the design, i.e., the function
2 before it licenses the implementation. So there ought to be
3 two separate licensing procedures.

4 These are the first two issues: Is it feasible to
5 produce correct software and what engineering practices?

6 I think it's feasible. I've said that and I've
7 said what I think we should do.

8 Passed out to you besides the slides was another
9 paper that discusses some more material and discusses the
10 mathematical ideas in somewhat more depth, which I'm sure
11 you may want to read.

12 Developing the shutdown system.

13 We should develop the software from black box
14 functions contained in an improved complete rigorous
15 specification.

16 Develop the software top down in a set of small
17 refinements.

18 Verify each refinement with rigorous mathematical
19 proof. You've got an intended function and you've got a
20 rule for every part and you verify that.

21 Record each refinement in the design hierarchy.

22 Then the complete design trail must present a full
23 and complete proof argument that the software will provide
24 exactly the behavior defined by the specification.

25 If humans were completely infallible, then we

1 would always have correct software as a result of this
2 process. If we really do this right with these small
3 software systems, I will be very surprised if that software
4 isn't correct before it's ever compiled.

5 Certifying the shutdown software. For each
6 software item we should begin with a development team and
7 satisfy the software is correct.

8 Then we are going to construct a statistically
9 significant sample, generate the test cases.

10 The certification team will establish software for
11 execution. Complete separation of responsibilities.

12 The certification team will then run sufficient
13 failure-free executions to enable the software to be
14 certified to the desired reliability.

15 If any failures are encountered, the software
16 should be rejected if the fault causing the failure is not
17 the result of some simple mathematical fallibility. Any
18 failure due to mathematical fallibility should be fixed by
19 the development team.

20 Then we have to do the same thing for the shutdown
21 system where we are now looking at all of these components
22 together. We've talked enough about that.

23 The specification challenges. It is one thing to
24 develop software in accordance with the specification. We
25 also have to develop the specification. So we have to

1 specify the desired behavior for the shutdown channel.

2 That's from beginning to end.

3 Then we have to ensure the specified channel
4 behavior considers all safety-related possibilities. That's
5 a very, very important ingredient, but that's a
6 specification function.

7 Then we want to allocate a mission to each of the
8 channel components, whatever they are.

9 Then we want to prepare the specification for each
10 defined component's behavior.

11 Then we want to verify that the composite behavior
12 of the behavior specified for each of the components is
13 equivalent to the specified behavior for the shutdown
14 channel.

15 We can now do a verification right now that the
16 mission, the specification that has been allocated to each
17 of these components, that the composite behavior of those is
18 equivalent to the behavior that you want for the whole
19 shutdown channel. That comes from just combining black box
20 function.

21 Then we've got a select or build components to
22 minimize the possibility that a common mode failure
23 overrides the redundancy built into the channel.

24 What is interesting about this list of things,
25 which is really the real heart of the matter, is that there

1 is no difference in this list if there are software
2 components in there or not software components. We should
3 do exactly the same thing. And we must do exactly the same
4 thing. There is no difference at this level between
5 software digital systems and non-digital systems in terms of
6 the activities of specification.

7 Specifying correct behavior is the same whether we
8 have software components or not.

9 All safety issues must be clearly identified and
10 mitigated by the specified design.

11 The specification task is considered more
12 manageable by knowledgeable authorities if software is
13 included due to the ability to specify improved logical
14 control logic and its dependability since it cannot wear out
15 prematurely. The only reason we go to software is because
16 we can get better control function, supposedly.

17 Then we want to use black boxes to specify the
18 desired behavior, and our next slide says something about
19 that.

20 Then we want to record the specification for the
21 system, for each component in an appropriately modified
22 cleanroom specification.

23 A black box function, as I'm sure most of you
24 know, is a function that defines a response in terms of a
25 sequence of all the previous stimuli that this component has

1 received, the sequence of those stimuli, and that's what a
2 black box function is. So a black box function is just a
3 mechanism that accepts stimuli for each stimulus, produces a
4 response before accepting another stimulus. Furthermore,
5 each response is uniquely determined by the history of
6 stimuli accepted by the black box. It defines external
7 behavior. It's the user's view of the system.

8 There is no state data, no internal inventions,
9 which is one of the things that is tough about particular
10 software. There is no procedure. It's a simple,
11 straightforward mathematical function that applied
12 mathematicians have been using for the whole 20th century
13 and it's just finding its way into programming.

14 Cleanroom specification. In the paper that I
15 handed out there is more argument about this, but we think
16 of it as being six volumes.

17 We have a mission;

18 A precise statement of the requirements;

19 A user's reference manual, which has to define the
20 stimuli and responses that have been invented for the system
21 to fulfill its mission;

22 The black box function; a validation argument, an
23 argument that argues that the system as defined by this and
24 this will in fact do this;

25 The Markov model, which we've talked a lot about;

1 And then a construction plan. In this particular
2 case that's fairly simple, because I think most of these
3 systems are so small they can be constructed in one rule.

4 You earlier made a comment about compilers.
5 There's a big question about whether you should have any
6 compiler at all involved in this particular software
7 development because it's may not be necessary and may
8 introduce a source of failure, as you pointed out.

9 Common mode failures seems to be a big issue. In
10 some of the material that was sent to me by the NRC there is
11 a lot of discussion about common mode failures. That's an
12 important issue, to design a system that is immune to common
13 mode failures. From a slide that we got from the NRR that
14 was the definition that was there. We prefer to say the
15 concern is that a hardware specification failure, a hardware
16 manufacturing failure, a software specification failure or a
17 software implementation may result in a common mode or
18 common cause failure of redundant equipment.

19 How do we handle that? In the material that I got
20 from you all it said the defenses against common mode
21 failures are quality and diversity. For software components
22 we get quality by using rigorous methods to develop software
23 and we develop and record the complete proofs for the
24 software and then we certify the software. We are going to
25 get quality from that.

1 We can get diversity if that's necessary. We can
2 develop and verify and certify a separate rule for each of
3 the different components that are redundant. There are an
4 infinite number of correct rules for any function. So I can
5 develop a separate rule for each different component so I do
6 get diversity.

7 Defense against software and hardware
8 specification failures. No matter whether you are using
9 hardware or software function, you want to use black box
10 functions. You want to develop a complete specification,
11 and then you want to review the specification with a jury of
12 peers.

13 Diversity. I'm not quite sure how to do that. I
14 haven't got an idea about that. And about hardware
15 implementation failures, I don't know about that either.

16 MR. LEWIS: In terms of software implementation,
17 if you have a complete proof, why do you want diversity?

18 MR. COBB: I'm not sure that you do. I thought I
19 used the word "if." I'm not sure you want that. You
20 certainly don't want it for economic reasons. In your
21 opening statement you said something about the NRC is prone
22 to be -- you didn't say overcautious, but whatever you said.
23 If you want to do that, it's possible to do from the same
24 function. The difficult issue here is specification.

25 MR. LEWIS: The 767 I was on last night coming

1 here did not have a jet on one side and a propeller on the
2 other.

3 [Laughter.]

4 MR. COBB: That's very good. I'll remember that.

5 The question is, is it feasible to specify a
6 correct function and what engineering practices? I argue
7 that if it's possible to define when a power plant is to be
8 shut down because it's entering an area of unsafe operation,
9 then it must be feasible to specify a system that will
10 perform the desired mission, including minimizing the
11 probability of common mode failure. If the physicists can
12 do this, we ought to be able to develop a system.

13 The designer should make use of black box
14 functions. The specifications should include these.

15 An important recommendation is that the Nuclear
16 Regulatory Commission should license the design prior to its
17 implementation. The implementation can then be licensed
18 when it's demonstrated that the implementation provides the
19 specified behavior. Because it's the design that is
20 crucial, not the manufacture.

21 MR. CARROLL: Would you be surprised to learn that
22 that is exactly what they are doing in the case of these new
23 plants?

24 MR. COBB: No, I wouldn't be surprised, because I
25 think that is good wisdom. I think that that is a very good

1 idea. But no, I did not know that. Nobody has told me.

2 In a more complex case, if I have four black box
3 functions defined here, the concept of being able to
4 mathematically derive the composite behavior of C1 followed
5 by C2 followed by C3 followed by C4 and derive the function
6 for the whole shutdown channel is an important idea. Then
7 we can compare the derived black box function with the
8 intended black box function. Then we know we've made a
9 proper allocation of specifications here so that this
10 followed by this followed by this followed by this will in
11 fact do what we want to do here.

12 The first person, to my knowledge, that wrote
13 about thinking about how to do this was Gabriel Crone when
14 he was at GE Research in the late 1930 and early 1940s. He
15 tried to reason about this. His papers are difficult.

16 Then the next person who I know made significant
17 contributions to this area was George Danzig, who is at
18 Stanford now, was at Rand. But the person who really got
19 this all right was Harlan Mills in the Mills-Linger-Abner
20 book which was published in 1987, I believe.

21 That's a very important mathematical property of
22 systems that can really be exploited.

23 The feasibility of the economical development.
24 Most of the required effort is necessary even if the system
25 does not contain software. The inclusion of software

1 components causes concern for two reasons.

2 The hardware engineers have had a history of
3 success, so we don't worry about their ability to design
4 good system.

5 Software developers have had a history of
6 confusing specifications for preparation of software
7 development. They've also had a history of using inadequate
8 practices. In summary, we've not got a great track record.
9 But I think using these practices software developers can
10 efficiently produce quality software.

11 At least one place that we are working now
12 developing software to go on boards that are going to
13 control devices there are hardware people working and
14 software people working. In this particular installation
15 the software people are producing higher quality designs
16 than the hardware people. It's the first time ever in this
17 particular company. That's fantastic.

18 They are feasible, they are economical, and they
19 work.

20 I think I've said everything on this slide. You
21 can read it. I wanted to have this text so you could read
22 it if you were interested. We've summarized that.

23 One of the things that I was asked to say
24 something about by your staff was there was a proposal that
25 was, I guess developed by people at Livermore about a design

1 and appraisal cycle. I was asked to comment how I would use
2 these ideas.

3 From project invocation the software developer
4 should prepare a statement of the mission, a system
5 development plan and a software development plan. These
6 should not be separated. They should be separate documents
7 but they are an integral part of the same operation. And
8 then a quality plan.

9 Then the NRC can have a process review and
10 acceptance.

11 At the system design level we should have a
12 specification for the shutdown system; a specification for
13 each channel, which I understand in most plants they are
14 talking about four where you have got voting; and then a
15 specification for each component.

16 Then we can have an IV&V contractor that is
17 independently hired that can produce a process audit report
18 to make sure that these people are following this process
19 and an analysis of quality matrix which can be appraised
20 about this.

21 Then there ought to be a design approval and
22 licensing of that design right now of that software design.

23 Then we can have component development. We
24 develop the design and verification hierarchy, all the
25 refinement, all the proofs, and record them. They could be

1 published in a little book.

2 We have the component, we have a test bed, and we
3 have a certification report. The certification report
4 defines exactly the test history. The IV&V contractor can
5 do a process audit report and an analysis of quality matrix,
6 exactly how the manufacturer has behaved in doing this.
7 Then we can have a component certification and licensing.

8 Then we can put everything together. Channel
9 integration. We have the channel, the test bed
10 certification and a report.

11 Then we can have a process audit report.

12 Then we can have the licensing of the entire
13 situation.

14 I think that's a process that works. The focus is
15 on the specifications and then on the implementation and
16 then on the integration.

17 These are beautiful set point reviews for the NRC.
18 If people follow these rigorous practices they will not have
19 any difficulty. This process of proofs and certification
20 tests will find out any bad code. If somebody tries to
21 license a bad system, that's going to be found out.

22 The question is, would you license a marginal
23 system that you shouldn't have licensed, or would you reject
24 a marginal system? I think that is going to be at the
25 specification level, not at the implementation level.

1 These reliabilities and these reports, you'll have
2 to think hard about exactly what you have to show in order
3 to do this acceptance. Whether it's hardware or software,
4 it's got to be economically feasible. If you've got a
5 hardware system, it ought to be tested with the same Markov
6 model; it ought to be tested exactly the same way. There is
7 no difference whether I have a bunch of relays or a piece of
8 software there. They both can have logic failures.

9 For your convenience, here are the kind of
10 documents that I think should be produced for each of the
11 specifications. I think you have an idea what they are.

12 MR. KEMMERER: Where does your voting show up? I
13 didn't see it in the previous one and I don't see it in
14 here.

15 MR. COBB: I look at voting as just being another
16 component.

17 MR. KEMMERER: So on that slide that you just had
18 where is it?

19 MR. COBB: There would certainly be a
20 specification for the voting component. This is a
21 specification for the entire shutdown system. So that would
22 be a specification for this whole thing. This would be a
23 specification for this channel here which allocates mission
24 to this component, this component, this component, and this
25 component. To all of these different components. In that

1 particular case I think there are 14 components. I have to
2 have a specification for each component separately, and one
3 of those components is the voting component. That has got
4 to be specified just like any other component.

5 MR. KEMMERER: When I look at your high level
6 diagram I see the four channels as the first breakdown of
7 the system, the four channels plus the voting. When I saw
8 what you had on your diagrams there, you had the channels,
9 but I didn't see the voting.

10 MR. COBB: This is a channel. This is one channel
11 here. So voting is at the channel level.

12 I think we are having a communications problem in
13 terms of words.

14 MR. KEMMERER: Yes.

15 MR. COBB: Then I was asked to comment on the
16 methods the NRC should follow to determine if a license
17 should be granted.

18 For process review the plan should clearly
19 separate the specification preparation and the software
20 development.

21 The development method should be rigorously based
22 on a mathematical foundation.

23 The development method should minimize debugging.
24 I think it should eliminate it.

25 The process should be clearly identified.

1 The design and license specification should be
2 complete and include black box functions and usage profiles.
3 Otherwise the NRC should use the rules it currently uses for
4 other systems.

5 At the software level we want to evaluate the
6 process and failure history.

7 They would like to evaluate a sample of the
8 proofs, and if any proof is found in error, they should
9 reject the system, the software. You could do 100 percent
10 samples if you like. A 10 percent sample would probably be
11 adequate.

12 And then evaluate the certification history.

13 For the channel and certification licensing,
14 evaluate the process and failure history and evaluate the
15 certification history.

16 You really would like people to have a good
17 process, because imperfections are caused by the process.
18 They're not caused by anything else. So therefore you want
19 to have a process that is well defined, that is rigorous,
20 that makes engineering. You don't want to have an ad hoc
21 process, which we typically have today in software
22 development.

23 I think from an adequacy standpoint the proposed
24 procedures will identify systems and software that is
25 failure prone

1 The one thing I didn't say and haven't got a slide
2 on. If somebody wants to license software that they have
3 done all of this for, you can use function abstraction and
4 you can abstract out the function. You can ask them to do
5 all of these things retroactively. The software is likely
6 to be rejected because it's likely to be failure prone, but
7 if it's not, the system will find that out. What we want to
8 do is eliminate failure-prone software.

9 The procedures used should be geared to reject
10 marginally acceptable systems. To do this, it will be
11 necessary to prepare a detailed certification plan.

12 If the recommended software development methods
13 are followed the software could be -- I think it will be --
14 a very reliable element, because once it's working it
15 doesn't fail. It's not like a relay that can get stuck.

16 Relative to specification correctness, all human
17 activity is subject to fallibility. That is why effort must
18 be focused on the design specification and the design should
19 be subject to separate licensing action. It's the design
20 that's crucial and that's independent of whether we have
21 software or hardware in there.

22 The question that I haven't had a chance to really
23 think too hard about is what is the probability that the
24 regulators, the NRC, will make a licensing error?

25 We can have two kinds of errors. We could accept

1 a system that should be rejected and we could reject a
2 system that should be accepted. From the NRC's standpoint,
3 this is a very expensive proposition and this is not too
4 expenses, so you clearly want to bias it so that you have a
5 larger probability of doing this than doing this.

6 Then relative to specifications and relative to
7 implementation we should separate those two concerns.

8 I don't know what the likelihood of a design
9 oversight is. One of the things that Professor Leveson has
10 been really focussing the software community's attention on,
11 I believe, is that we have to get the specifications right,
12 but we have clearly have got to make a distinction between
13 specifications and implementation. Faults can come at this
14 side or faults can come at that side, and we've got to get
15 the specifications separated from the implementation.
16 Otherwise we have too big a problem; it's out of
17 intellectual control.

18 Relative to implementation, I think we have a very
19 low probability of accepting a system that should be
20 rejected. We want that to go to zero, but I think that's
21 very, very low.

22 We would like this to be a bigger probability
23 here, and we got to figure out how to make those two numbers
24 get exactly the way you would like.

25 The NRC and the manufacturer of these software

1 items has to focus its attention first here and then here
2 and make sure the specifications are right and then get the
3 implementation right. If you try to do the implementation
4 and the specifications all at the same time, you get into
5 the following kind of dilemma.

6 I was with an Air Force operation two weeks ago.
7 On one system that they have, which is an important system
8 for our national defense, they finally said this system must
9 meet the specifications because we'll make the
10 specifications equivalent to what we've got. That's not the
11 way to get a good system. Unfortunately, that's what we do
12 with a lot of software systems today. We declare it's done
13 because it's done. What we want to do is specify it and
14 then implement it.

15 That makes a whole lot of good sense. I know that
16 that is what other engineering professions have been doing
17 all along. We are only 40 years old, but that doesn't mean
18 that we can't get good.

19 I was asked on the fax that I got from you all to
20 comment on five topics. I'm only going to comment on four
21 of them because the fourth one I don't know anything about.

22 MR. LEWIS: That usually doesn't inhibit people
23 [Laughter.]

24 MR. COBB: The first question was, is the size and
25 complexity of the nuclear safety-related software

1 sufficiently limited to prevent effective use of formal
2 methods?

3 The proposed designs decompose software items into
4 small blocks each with a well defined self-contained
5 function. The software development, verification and
6 certification issues are all well in the bounds of what can
7 be expected to be accomplished by things like cleanroom
8 methods as they are understood and applied in 1993. The
9 current craft-based practices that are in widespread use
10 today are not acceptable for most applications to which they
11 are being applied and certainly not acceptable to the one
12 which is the subject of this hearing.

13 So yes, my opinion is that the size and complexity
14 is simple enough and it is essentially no different whether
15 there is a software component in there or a hardware
16 component.

17 What is the overall assessment of the best methods
18 for software certification/V&V in terms of effectiveness,
19 efficiency and comprehensiveness and degree of assuring high
20 reliability and fidelity in software system performance?

21 If humans were infallible the proof prepared
22 during the development would guarantee the software will
23 provide correct performance. I believe in this if the
24 manufacturer takes the right care that will be the case.

25 Since humans are fallible, subjecting the proof to

1 independent peer review will increase the likelihood of
2 detecting a fault. In this particular case they can be
3 subject to as much peer review as people would like.

4 The use of statistical testing will also provide
5 an added measure of assurance. That's the way you have to
6 look at this testing, not to test in quality but to confirm
7 the existence of quality.

8 Using the recommended methods the question will
9 not be, does the software behave as specified? The question
10 will be, is the specified behavior adequate to perform the
11 defined mission? Which gets you out of the software issue.
12 These procedures will get you high quality software.

13 Topic three was endorsement of specific software
14 design criteria as promulgated by professional society
15 standards. In general, I would say that software
16 development standards codify the craft-based practices and
17 therefore are insufficient for developing quality software.
18 So the method and practices we are recommending are
19 sufficient. They are certainly permitted by these standards
20 but lesser quality is also permitted by those standards, and
21 those lesser qualities, in our opinion, are not adequate.

22 I've skipped topic four.

23 Topic five, approaches to managing software risk
24 both in software development and safety system operation.
25 What we want to do is use the highest quality software

1 development practices possible.

2 Avoid all high risk activities. No arrays.

3 Arrays are hard to prove.

4 No dynamic memory management. Dynamic memory
5 management is hard to prove.

6 No multitude of data types. These simple
7 activities don't require any of these so-called
8 sophisticated programming techniques. The stress has to be
9 on getting a rule for the function that is very easy to
10 prove. Hard functions are hard to prove and you don't want
11 to get into those.

12 You want to carefully prepare and evaluate the
13 specifications from a safety analysis standpoint. Professor
14 Leveson's comments on that area are absolutely right on.
15 We've got to think about it from a safety standpoint, but
16 that's a specification issue.

17 Designers want software functionality to specify a
18 more robust system. Designers want software devices because
19 software components do not fail to operate as required as
20 hardware components often do.

21 Systems which contain software components can
22 provide self-assessment so the premature failure of hardware
23 devices can be detected as soon as they occur. That's a
24 very important ingredient in control system.

25 In general, software provides system benefits.

1 Craft-based development methods introduce
2 implementation risk; engineering-based development methods,
3 I think, mitigate software implementation risk.

4 In another decade or two as all software
5 development moves up to this higher standard, which it
6 absolutely will, we'll be questioning why people back in the
7 1990s were so worried about software.

8 In summary, I think developing a shutdown system
9 requires specifying the desired behavior for the shutdown
10 system.

11 Ensuring the specified behavior is complete.

12 Defining the components that comprise the shutdown
13 channel.

14 Allocating a mission to each of the channel
15 components: voting, analysis, all the multiplexer
16 components.

17 Specifying the behavior for each component.

18 Verifying that the composite behavior for all the
19 components provide the same behavior as the specified
20 function for the entire system. That's a very important
21 proof element.

22 Then developing each component by implementing a
23 specified behavior.

24 Documenting complete proofs of correctness.

25 And certifying each component provides the

1 failure-free behavior.

2 We have got to assemble the channel; then we want
3 to certify the channel provides failure-free performance.
4 We've already proved that if these are right, these are
5 right, and now we want to certify it.

6 Cleanroom methods provide the means to accomplish
7 6, 7 and 9, and cleanroom methods also provide the means to
8 support 1, 2, 4 and 5, the specification activities. But
9 you have to think hard. If you focus the thinking on the
10 specifications, then you're likely to get the specifications
11 right.

12 The only step that's different, depending on
13 whether you've got software components or hardware
14 components, is how you do step 7. All the other steps
15 should be required for a hardware or a software
16 implementation. The NRC should regard those specifications
17 as completely independent.

18 In conclusion, the components provide substantial
19 benefits.

20 The risk associated is that the delivered
21 components may include implementation faults that can result
22 in a failure-prone operation.

23 The software implementation risk will be mitigated
24 when software solutions are engineered instead of crafted.
25 The engineering methods we have proposed permit the industry

1 to develop, verify and certify failure-free software
2 components, and in this way shutdown channel designers can
3 obtain the benefits offered by the software components.

4 I think I'm about one minute ahead of time,
5 according to my watch.

6 MR. LEWIS: I am deeply impressed by your
7 midcourse correction.

8 All software does have the problem of bugs. One
9 bug that is afflicting us is that Nancy Leveson, who was
10 scheduled to speak after the break, has the flu. So after
11 the break what I would like to do is to ask all of us,
12 especially our consultants Pat and Dick, to open a
13 discussion on what Rich has said and the larger issues that
14 he has raised that perhaps he didn't address, and we will
15 make it a free-for-all between the break and lunch.

16 So let's break and reconvene in precisely 17
17 minutes.

18 [Recess.]

19 MR. LEWIS: Let's reconvene.

20 What I would like to spend the rest of the morning
21 on before our lunch break is mainly to get reactions from
22 anybody but especially from our consultants Pat and Dick
23 about what we've heard, and indeed about the larger question
24 that we are trying to confront, which is how to fold the
25 digital technology into the nuclear business hopefully

1 without any compromises to safety and within the context of
2 the existing system, both regulatory and mechanical.

3 Let's start with Dick and Pat. We'll invite
4 comments from the staff, the audience, from anybody.

5 This afternoon we are schedule to hear from the
6 industry on what their actual experience has been trying to
7 get into this business. I don't know to what extent that is
8 going to be complete disjoined from what we talk about this
9 morning. We'll find out as we go along. I hope we can keep
10 this a completely informal procedure. I've tried to set a
11 standard for that.

12 Dick or Pat.

13 MR. PLACE: I don't want to seem like I'm sort of
14 beating up on the cleanroom method but it raised some
15 questions in my mind. The penultimate slide talked about
16 the nine steps through the process of developing software
17 and step 8 was one that perhaps gave me concern, that was
18 assembling the channel.

19 One of the problems that came to my mind while
20 listening to the talk was that if we specify the behavior of
21 the system, we can make some statement as to what it's going
22 to do and how it's going to do it. We can specify the
23 behavior of the individual components. We were told that we
24 can and indeed I believe we can. We can show that the
25 composition of the specifications of the behavior of the

1 components meets the requirement laid down by the
2 specification of the system as a whole.

3 We can show that the components in fact satisfy
4 the specification for each component. So we could assume
5 perhaps that by assembling those components we will have a
6 system that does in fact satisfy the expected behavior of
7 the entire system.

8 My concern is that when you assemble components
9 sometimes they have interactions between themselves due to
10 the implementation that were not observed and have not been
11 observable. It could be as simple as memory overlap, and so
12 we get an implementation that doesn't meet the specification
13 for the entire system.

14 It seems therefore that you've got a requirement
15 to demonstrate independence of components. I don't think
16 that was addressed this morning. So I raise that as my
17 first issue.

18 MR. COBB: You're right. We talked about that
19 during the break. We didn't address independence of
20 components. Step 9 is trying to certify that you do have
21 that independence.

22 I think the easiest way in this particular case to
23 prove that you have independence is that each of the
24 different software components are to be on a processor
25 itself. From my conversations with the proposed

1 manufacturers of plants, each one of these functions would
2 be on a sperate processor and not on the same processor. So
3 you are generating complete independence of components.

4 If you are going to put them on the same
5 processor, then you do have to have an additional proof so
6 they can prove that the implementation is independent.

7 A 386 processor costs maybe \$50 or less. So the
8 idea in these shutdown channels to actually have physical
9 hardware separation so you really can assure an arm's-length
10 transaction is a very important part. From my minimal
11 conversations with people who are talking about designing
12 these systems it was my understanding that they already have
13 that in mind and are talking about separate channels.

14 If somebody came to you and tried to license a
15 channel that didn't have this separation, they now have an
16 added degree of burden of proof and they are going to have
17 to have a little bit more complex operating system in order
18 to do the memory management and things like that. One of
19 the beauties of the design as I understand what these people
20 are talking about is that they have eliminated that as a
21 source of concern in their design already. That just
22 changes the element of proof if you've got to do that.

23 You're absolutely right. I've assumed that that
24 was the case because that's what I was told was the case,
25 but I didn't make that assumption clear in my talk.

1 MR. PLACE: Then it seems like you've now shifted
2 the burden onto the communication mechanism as being another
3 source of failure. The communication between processors is
4 now yet another source of failure that you have to
5 investigate.

6 MR. COBB: Yes. That's a hardware element which
7 has to be considered by itself. It's a physical wire
8 between boards. There's no question about that. But that's
9 stimuli and responses. I don't think that would be too big
10 a burden.

11 MR. PLACE: But usually there is some software on
12 top of that which also manages the communication. This is
13 where the system gets complex one way or the other.

14 MR. COBB: The communications has to be part of
15 these two systems. It absolutely has to be part of these
16 two systems.

17 The point is we can make a bad design, but what we
18 want to do is make a good design. Regarding software as a
19 mathematical proposition, I'm assuming that the design that
20 we are implementing the software for is a good design. We
21 as humans are able to screw up anything if we try hard
22 enough but we ought to try to make it good.

23 MR. LEWIS: Paul, did you want to say something?

24 MR. SHEWMON: Yes. I want to bring up something
25 that is tangential to that. I believe we will come later to

1 the staff and the industry.

2 It has to do with the separation between safety
3 systems and control systems. I don't know your background
4 in the nuclear systems, but it's sort of a hoary old one.
5 It says you'll have a few simple circuits and if these go
6 out of spec you'll scram the reactor. Then completely
7 separated and isolated from this you'll have something that
8 will worry about control the rest of the time. As we have
9 got computers into these I have the impression they have
10 merged some.

11 One advantage that I've heard from a separate
12 safety system is that if you have just a few inputs and a
13 few conditions you have to meet it is much easier to check
14 out if it is indeed separated. If indeed now it becomes all
15 part of one wonderful computer system, then there are a lot
16 more interactions.

17 I guess my question for later in the day is, where
18 are we now with regard to the separation of safety and
19 control systems? Has that gone by the board in new plants?
20 If it's a red herring you can say we'll take that up this
21 afternoon.

22 MR. LEWIS: No. I'm only wondering why you are
23 looking at me for the answer to that question.

24 [Laughter.]

25 MR. SHEWMON: I'm looking at you for the one to

1 say that's out of order.

2 MR. LEWIS: No, no. It's not out of order. I
3 would think the staff would respond to that. I'm not your
4 staff.

5 MR. STEWART: My name is Jim Stewart. I work for
6 the Instrumentation and Control Systems Branch.

7 For all of the current designs we have right now
8 the computers for the safety and the non-safety systems are
9 completely separated.

10 MR. SHEWMON: So any software written for the
11 safety systems then would have the more limited set of
12 commands.

13 MR. STEWART: The safety computers are physically
14 different computers with physically different software than
15 the non-safety computers.

16 MR. SHEWMON: And you do your best to make sure
17 that they are indeed separate and can't influence each
18 other?

19 MR. STEWART: Yes.

20 MR. SHEWMON: Fine.

21 MR. COBB: I would like to comment on that. I
22 have commented on quality of software. It would seem to me
23 that if the software that is providing plant control is also
24 of higher quality and it has less failures, then you are
25 better off, because then you're going to get into these bad

1 conditions less often. So that seems to me to be a good
2 thing.

3 I would encourage anybody who is developing
4 software to try to develop the highest quality software they
5 could. As I understand it, the difference between the
6 software that should be in the safety shutdown systems and
7 maybe the control systems is not the level of quality but
8 the level of proof of quality. On the shutdown system I
9 want to be able to prove to the whole world that it's
10 perfect whereas with the control system maybe I don't want
11 have to prove it to the whole world; I just want to be sure
12 myself that it's right. I think it's the standard of proof
13 that's different, not the quality requirement.

14 MR. SHEWMON: The more options you've got coming
15 in the harder it is to check things out.

16 MR. COBB: That's why it makes the proof harder.
17 That's why you don't necessarily want to have that big high
18 standard for that.

19 MR. GALLAGHER: I am John Gallagher in the ICSB.
20 There has just been an International Electric Technical
21 Commission standard issued that deals with this and has
22 great requirements based upon the level of importance to
23 safety. So as you say for the safety systems, it requires a
24 very high level of assurance. As you move down to control
25 or monitoring systems, then it's graded down, but there are

1 still quality requirements that have to be met for the
2 purposes that you just stated.

3 MR. LEWIS: Tom.

4 MR. KRESS: I have a question for Mr. Cobb. In
5 your talk I had a little trouble figuring out in your Markov
6 chain how you would arrive at the correlation between
7 states. I didn't see any real way of having the knowledge
8 ahead of time to determine the probability of going from one
9 state to another. Could you expand on that just a little
10 bit? I had a lot of difficulty trying to figure out how you
11 would develop those correlation functions.

12 MR. COBB: The beautiful part about a Markov model
13 is that the only probability we have to estimate is the
14 probability that if we are in a particular state that we
15 will move to another state. If you have a row and you are
16 at state I and there might be ten states that you can move
17 to, you have to estimate what the probability of moving to
18 each one of those ten different states is. Or if there are
19 only three, each of those different states. Is it a third,
20 a third and a third, a quarter, a half and a quarter?

21 MR. KRESS: To be specific, are these states
22 physical states of the system like a temperature and a
23 pressure and a power and an activity level?

24 MR. COBB: Right. I haven't developed the states
25 for any particular shutdown systems.

1 MR. KRESS: You know if you are in an upset
2 condition that you want to do something with this software.
3 You know that the temperatures and pressures are likely to
4 both be rising at the same time.

5 MR. COBB: That's exactly how you have to estimate
6 these probabilities. I haven't determined how to estimate
7 the probabilities for any particular system. I have had
8 some conversations with Leo Beltracchi of the NRC. He asked
9 the question, is it possible to estimate all of these
10 probabilities that things can happen from the standard data
11 that is reported from current nuclear power plants?

12 I've never given him a real satisfactory answer
13 because that's an answer that is going to require more work
14 than 15 minutes or an hour conversation. I suspect that the
15 answer is that all of these data are available.

16 I think the more important answer from your
17 standpoint is what we want to do for the shutdown systems is
18 we want to bias the probabilities to test the strata where
19 the plant is going to be close to the state of out of
20 control. The plant is going to be in control for a large
21 fraction of it's life, maybe three months in a row. If we
22 try to do tests that last three months, we might only get
23 one failure condition if we use the normal sort of
24 probabilities.

25 I think the usage that we are interested in here

1 are the usages that are going to tend to cause the plant to
2 go out of control. Presumably you know a lot more about the
3 physics that go on inside these plants than I do, because I
4 know nothing.

5 MR. KRESS: I know a little more than that.

6 MR. COBB: That's the reason I assume you know
7 more.

8 What we have to do is develop the right function
9 and the right conditions. I'm confident you can do it. I
10 just don't know how you do it. That's an engineering
11 problem.

12 The important point is that with these software
13 systems and these systems we can generate constructively any
14 complex set of usages where the memory develops in the
15 software and generate these memories by using this
16 stochastic process, which is only a Markov process, where
17 the original probabilities only demand that we know which
18 state we are in, that we don't have to remember that we got
19 to this state from this state, because the software will
20 build that up internally as we generate the test cases.

21 In the appendix there are two or three pages that
22 argue from a mathematical standpoint exactly why this is the
23 case.

24 How you are going to estimate these I don't know,
25 but you've got a wealth of data out there about how nuclear

1 reactors are supposed to work and you have a large
2 theoretical collection. So I assume that people know these
3 things and we just have to transform them into a reasonable
4 model. You should do that independent of whether it's
5 software or hardware. It makes no difference.

6 MR. LEWIS: One of the classic issues along these
7 same lines in terms of which state follows which state is
8 the possibility of getting into a loop. The example that
9 jumps to mind, which is not a good one for this context but
10 is in other computer contexts, is a feedback shift register.
11 You can run it, test it, do everything you like in it, but
12 if there is any fluctuation, a lightning stroke or something
13 like that ever sets all the bits to zero, then it will stay
14 that way forever and ever. Normally you would avoid that
15 state. But it's an example of a case in which a state which
16 is not a normal state of the system may turn out to be a
17 loop to hang up the system.

18 I don't quite know how you would track down things
19 like that by looking at the history of normal usage of a
20 system. That one is easy to track, because you see it in
21 front of you when you write the thing down.

22 I worry about the system being thrown into
23 completely unexpected unexperienced states by other events,
24 perhaps external events, which you have not analyzed. Short
25 of a formal proof that you've covered the entire function

1 space, which of course is the ideal, I'm not quite sure how
2 you deal with that.

3 Since I've mentioned the question of formal V&V,
4 we haven't given Dick a chance to say anything and we
5 probably should.

6 MR. KEMMERER: I find this very interesting. In
7 reading the few things that you sent to me I noticed that
8 this is the ninth or tenth in a series of meetings. The
9 question you asked is exactly the question I had. I'm
10 surprised, because I thought maybe you had all this
11 background information.

12 I think the model of use distribution is the thing
13 that bothers me the most. It bothered me about cleanroom
14 when Harlan originally was doing it, because there was no
15 unit testing. The idea of no unit testing was that when we
16 get the system out there we're going to do system testing
17 based on this random selection based on our knowledge of
18 use. I remember Hal giving a talk at Santa Barbara quite a
19 few years ago after Chernobyl, telling us about the
20 combination of things that made Chernobyl happen.

21 MR. LEWIS: I don't remember that.

22 MR. KEMMERER: That's okay. I'm younger than you
23 by a year or two.

24 What I am concerned about are exactly those
25 things. When we find the real problems that are

1 catastrophes it's things that don't occur very often at all
2 and may not have occurred at all in the past.

3 Did you want to address that?

4 MR. COBB: I'd be happy to address it with the
5 little wisdom I have.

6 That's a specification issue. I absolutely agree
7 that if we are going to develop any kind of system you
8 really have to worry about finding the boundary of the
9 domain and then you have to worry about what's going to
10 happen when you go outside of that domain. The only thing I
11 can say is that, yes, you have to think about all of these
12 instances and you have to try to define them. If we are
13 going to have any systems operating in the real world, we
14 have to think hard about all of these things that can happen
15 that we don't expect to happen and then build in safeguards
16 against them, whatever that means.

17 On the other hand, we want to keep the system as
18 simple as possible so that we don't over complicate the
19 thing.

20 I think by thinking formally about the
21 specifications as mathematical functions it enables you to
22 think about these things a great deal and it enables us to
23 communicate about what we're really talking about.

24 I think when you are in the realm of all of those
25 kind of mathematics that I don't completely understand I

1 think that it's impossible to prove that a specification is
2 completely correct, that it will anticipate all situations.
3 I think that's one of these things that's unprovable. We
4 are faced with that problem. That's independent of
5 software, hardware or whatever it is.

6 The cleanroom ideas, the formal ideas for
7 developing software accurately in conformance with the
8 specification let you focus your attention and energies on
9 getting the specification right as opposed to getting the
10 implementation right.

11 I can't tell you you are going to get the
12 specifications right, because I don't know. The only thing
13 I can tell you is that whatever specification we think is
14 right we ought to be able to implement, which is not
15 necessarily true in all software today.

16 MR. KEMMERER: I'm a proponent of formal methods,
17 so I agree with what you say.

18 You said in your talk is you are stressing the
19 importance of getting the specifications correct. If Nancy
20 were here, one of the things I'm sure she would remind us of
21 is the -- you brought up the shuttle flight software. IBM
22 claims to have zero defects. Zero defects is a term they've
23 used. I didn't make that up. On the other hand, in May of
24 1992 there was the Endeavor flight with the problem with the
25 Intelsat. In that case, if you read Aviation Week in July

1 of 1992, Ted Keller from IBM says, well, the IBM process
2 didn't have an error because the specifications were wrong.

3 It stresses your point that the specifications are
4 very important. I think that's where you have to put your
5 money.

6 MR. COBB: The whole point is that in software up
7 until recently, and even now only in the very good shops, it
8 has been so hard to implement software. We've been focusing
9 our attention on doing the implementation and sort of
10 forgetting the specifications. You can't forget the
11 specifications.

12 I think what I did try to stress in my talk a
13 little bit was that you have to separate specification
14 failures from software failures, because the way you
15 eliminate software failures is different than the way you
16 eliminate specification failures. If you try to confuse
17 them, then it's very hard to think when you think about too
18 many problems at the same time.

19 I am very much in favor of separating
20 specifications. In this case the issue is specifications,
21 not implementation.

22 MR. LEWIS: On the other hand, it's important to
23 keep roles and missions separate. We are, for example, an
24 advisory committee to the Nuclear Regulatory Commission.
25 People sometimes think we actually do things, but we don't.

1 All we do is advise. The Nuclear Regulatory Commission in
2 turn is a review commission, a regulatory commission. It's
3 easy to fuzz the distinction between reviewing somebody
4 else's work and getting it right in the first place, getting
5 the specs right in the first place. The NRC isn't in the
6 business of writing the specs; it's in the business of
7 reviewing the software specifications.

8 So I'm not quite sure whether in order to do its
9 job it's important for the NRC to insist that the people who
10 come in for review have their specifications clearly and
11 unambiguously stated, which everyone would think is a good
12 thing to do, and how the NRC is in turn going to assure
13 itself that that has been done, that there are no internal
14 contradictions of the specifications and that kind of thing.

15 I'm trying to keep the designer's role separate
16 from the regulator's role and I'm having a little bit of a
17 problem.

18 MR. KEMMERER: I think that is actually one of the
19 problems with the cleanroom approach, that if you were using
20 a formal specification verification approach where you had
21 tools to back you up. One of the advantages for NRC is that
22 they can run it through the theorem prover to check those
23 proofs again.

24 I'm not trying to put down cleanroom by any means.
25 It's had a lot of successes, but it is one way of checking.

1 On the other hand, I don't know the solution to
2 that you've got the right spec. You said it doesn't have
3 any inconsistencies. I think the inconsistencies are easier
4 to find. I think the main thing that we are missing are the
5 missing states, the missing statements about possible
6 situations. That's where I worry about this use
7 distribution.

8 MR. LEWIS: I understand that. I used internal
9 inconsistencies just as an easy example. I agree the
10 missing states are important. I'm having a little bit of a
11 problem visualizing what I would advise the NRC to be doing
12 to be able to assure the public that the plants are
13 adequately safe without in a sense redoing the spec writing
14 for the utilities that are coming in. That isn't the NRC's
15 job.

16 I keep beating on the NRC to hire more people who
17 have the capabilities to run things through the theorem
18 provers and do the computer science type things. They say
19 they are working hard, but as a matter of fact I did pick up
20 an NRC recruiting brochure when I was in Florida last week
21 and I didn't find any mention of computer science or even
22 electronics in their recruiting brochure. So the efforts
23 are well concealed.

24 Short of the kind of expertise that is represented
25 around the table, I'm not quite sure how they are going to

1 be doing these jobs.

2 MR. KEMMERER: Wasn't that the peer review that
3 you asked for in your process steps?

4 MR. COBB: Yes. I think that that peer review is
5 a very important ingredient.

6 To really do the specification, I think the first
7 thing you've got to do is insist that the specifications be
8 rigorous and mathematical. You've got to follow the kind of
9 things that Dave Parness talks about. You've got to really
10 think about the right form for the function is a black box
11 function because it's got no internal details; it's all
12 external details; so people could reason about it. You've
13 got to make the specifications public.

14 Yes, I think that the NRC ought to pay rigorous
15 attention the specifications. It ought to be up to the
16 manufacturers to come in and say that these are the
17 specification, this is our statement of the specifications,
18 this the function we are going to implement, this is why we
19 have this function, this is why we think that the domain is
20 the way it ought to be. The same way we do when we reason
21 about the stress analysis for a wing on an airplane. That's
22 exactly what we do. I think that has to happen. Whether
23 people need more computer science specification to do that,
24 you need all sorts of types.

25 If I was the NRC I would have hired a nice j. / of

1 peers and the manufacturer comes in and presents the
2 arguments to them about why they think this is the case and
3 people can ask questions. I think people by and large are
4 of good will and try hard to get things done right. It's a
5 very difficult issue.

6 I guess I've said this four times now. The
7 specification issue is completely independent of whether we
8 have software or hardware in this channel. There is no
9 difference at the specification level.

10 MR. LEWIS: There is no harm in saying something
11 four times. Lewis Carroll says three are enough, but
12 sometimes four are helpful.

13 Dick, it has been pointed out to me that there is
14 nowhere on the record what you mean by formal
15 specifications. Some people might think it means sitting at
16 a computer in tuxedo and tails.

17 [Laughter.]

18 MR. LEWIS: It would be good to put on the record
19 what you mean by that.

20 MR. KEMMERER: One of the things that came to mind
21 while Richard was just talking was the fact that one of the
22 things NRC ought to do is think about education for their
23 folks to find out just what formal methods are. Formal
24 methods are using mathematical techniques to express what it
25 is that the program should do. I thought Richard said this,

1 but maybe I'm hearing things because I'm on the other end of
2 that.

3 Instead of using English statements we are using
4 mathematical expressions and we are using things like first
5 order predicate calculus, which just made the guys behind me
6 probably fade away.

7 Is first order predicate calculus okay?

8 MR. STEWART: Some of us are into it.

9 MR. KEMMERER: I've spent a lot of time with DoD
10 in the security community. This is sort of deja vu, because
11 about ten years ago we were going through that with DoD,
12 saying you ought to use formal methods. As soon as you get
13 up and start presenting and put an upside down A or a
14 backwards E for a there exists or for all, half the people
15 leave the room. On the other hand, I think if you do
16 appropriate education you can say this isn't so bad. I
17 believe that the people who are designing these systems are
18 smart people. They are smart engineers and they shouldn't
19 be scared away by the logicians, which is what often
20 happens.

21 I didn't answer your question, I don't think.
22 What we want to use is a mathematical expression to say what
23 is the function that the system should satisfy. What this
24 also requires is that the implementation language is also
25 formally defined. Where we have an assignment statement we

1 have defined exactly what the effects of it are.

2 It's the only way that you can make this work in a
3 machine oriented way as opposed to manual proofs, which are
4 error prone. So we have to say for an assignment state or
5 an IF THEN ELSE exactly what the function is of that
6 particular statement.

7 By doing that we just do mathematical proofs in
8 the same way that we normally would do proofs like you did
9 in junior high school. Or high school or wherever you did
10 that. It's not as long as it was for you, Hal, but it has
11 been a long time for me too.

12 MR. LEWIS: It has been even longer for me, as you
13 pointed out.

14 I recently read the GAO report on the C17
15 software. I don't know if you have read that yet. Coming
16 down to grips at a DoD level now, it turned out that the
17 software written for the C17 was written as I was able to
18 count in seven or eight or nine different languages and
19 tendered for eight or ten different processors and computers
20 interacting in a way which was determined by cooperative
21 agreements among the contractors for the various subsystems.
22 It's a miracle that the airplane flew at all under these
23 conditions. It's a long way from that to the kind of
24 integrated development of software specifications that we
25 are talking about here.

1 In the nuclear business we do have the advantage
2 that the reactors are for the most part developed and
3 designed by a single vendor. Although the implementation
4 comes from many people, the design tends to come from a
5 single vendor, so there is a possibility of doing a more
6 integrated version of the software than there would
7 otherwise be. But still the problem of subcontracting and
8 fragmentation, as exemplified perhaps in its worst by the
9 C17 case, is around with us to.

10 I'm a theoretical physicist, so I would like to
11 see clean things too with clear specifications. I'm having
12 trouble visualizing it in this miserable and imperfect world
13 we live in. But that's an expression of my own problems.

14 Let me ask our consultants in general. I guess
15 Rich has told us what he would advise the Nuclear Regulatory
16 Commission to do. It comes fairly close to advising them to
17 design reactors but it doesn't quite get there. What about
18 the other two of you, Pat and Dick? If you were in our
19 shoes advising the Nuclear Regulatory Commission, what would
20 we tell them to do now?

21 MR. PLACE: I'll take a crack at that. It seems
22 that the Nuclear Regulatory Commission sets requirements on
23 reactors. They state what it means for a reactor to be
24 safe. They need to formalize their requirements of safety.
25 Then there is an opportunity to say, well, we can take a

1 formalized set of requirements or specification for a
2 shutdown system to demonstrate that that satisfies our
3 specification for safety. If it does, then we can start to
4 believe that we have a system that will in fact perform at
5 least as we would require it to perform.

6 The big problem is, what does it mean to be safe
7 and how can you formalize the requirements of safety? I
8 think the work that Nancy Leveson has done take us some of
9 the way along that route.

10 MR. LEWIS: One of the questions that comes to
11 mind is that the Nuclear Regulatory Commission has
12 promulgated a set of standards for the safety of reactors
13 and they are stated in a very formal quantitative way: there
14 shall be less than .1 percent of the risk of cancer for a
15 person near a nuclear power plant than his or her risk from
16 other sources. Translating that into a spec on the software
17 poses problems of not only doing the software right, but
18 also evaluating how right you've done it in a probabilistic
19 sense, which may not be appropriate for software evaluation.
20 How do we approach that?

21 MR. PLACE: Indeed, I think there is a huge
22 problem of assessing reliability or assessing safety,
23 because we can't afford to get real data. We get real data
24 by failures. and we can't afford that.

25 MR. LEWIS: That's a problem that isn't just a

1 software problem. That's a nuclear problem.

2 MR. PLACE: That's a problem in general.

3 We also have a problem where we expect very high
4 reliability. How can we assess high reliability if we have
5 to measure 10 to the minus 9 , which is a figure that is used
6 in the aircraft industry? It takes a long time that we have
7 to observe systems and see no failures before we can start
8 to believe that we may have the design level of reliability.

9 I think those are issues that need to be addressed
10 and probably can't be addressed by current technology. We
11 saw in the ACM conference in 1991 that assessing reliability
12 is very hard to do. In fact, I think Rickie Butler showed
13 quite successfully that it was impossible to do by current
14 technologies, that we just don't have the time to do these
15 assessments.

16 MR. KRESS: If you had to have that sort of
17 reliability out of the software, then I don't think one
18 would implement it. I think the reliability of the system
19 is going to be dominated by something else. So you really
20 don't have to show that kind of level.

21 MR. PLACE: I think there are other ways to reduce
22 the requirement for reliability. One of the other
23 suggestions I've heard is to reduce the amount of memory
24 that your system has. We heard from Dr. Cobb that one of
25 the big problems is the amount of memory that accumulates in

1 the system. If we can reduce that by perhaps resetting the
2 system on some periodic basis, yes, we run the risk that we
3 need bits of the system while it's in its reset mode, but on
4 the other hand we reduce the amount of memory that exists
5 within that system so it doesn't have the time to accumulate
6 a long history of failures.

7 We see examples of that in the Patriot missile
8 system. They say reset that every 19 hours, otherwise
9 you'll have a problem. Of course on one occasion they
10 didn't reset the missile system and there was a problem with
11 drift. That might be an area where we can improve safety
12 with reduced reliability requirements.

13 MR. COBB: I'd like to amplify that point. You
14 want to avoid functions in your specification that look like
15 infinite functions at all possible cost, because that's when
16 you accumulate memory. That's one of the things I seized
17 upon in this particular design where we have these four
18 channels. You are doing voting across these four channels
19 that people are talking about. What that means is that I
20 can actually reset one of the channels at a very frequent
21 interval and without impairing operations. I can do that
22 very quickly. Then I'm really invoking the software all over
23 again.

24 The point is whoever has thought up this four
25 channel design and all of the things that they are talking

1 about, they've really thought of some very clever ideas to
2 try to make them so that they are easier to verify as being
3 safe. One should give people credit for having thought of
4 good ideas.

5 That point that he just made about doing that, I
6 did look at the software that they had up at Darlington.
7 There the function that they had going in was an infinite
8 function. In terms of the specification for that software
9 system up there, they tried to everything they could to try
10 to make it as hard on themselves as they possibly could.
11 They didn't try to help themselves at all in terms of the
12 design. I was going to say they were naive. That's
13 probably an overstatement.

14 MR. LEWIS: It was the northern climate.

15 MR. COBB: It was five or six years ago that they
16 started that. That's a large fraction of the half life of
17 the software industry.

18 MR. LEWIS: In this country we're pretty good at
19 stretching things out too. It makes gainful employment for
20 engineers.

21 Pat, you mentioned 10 to the 9th. It's awfully
22 important when one uses numbers like that to know what one
23 is talking about. I'm fond of pointing out that at this
24 very moment back in Santa Barbara my computer in my study is
25 churning away on some miserable problem I gave it to keep it

1 busy, keep it out of mischief. It's doing something 33
2 million times per second. That's the clock speed. Each
3 time it does something it does something to 32 bits, because
4 that's the bus, and therefor 10 to the 9th things are
5 happening every second. I don't expect it to have a failure
6 every second. In some miserable and primitive way it has a
7 reliability that is far greater than 10 to the minus 9. On
8 the other hand, we don't count that as a reliability.

9 So it's important to define. In fact, I will tell
10 you that it failed the other night because the power went
11 off in my house and stayed off longer than my
12 uninterruptible power supply could keep the silly thing
13 going. So there are other things that intervene at that
14 level.

15 But 10 to the minus 9th doesn't mean anything
16 unless you know what you are talking about.

17 MR. PLACE: I'm not talking about the number of
18 operations per second that have to work correctly. One
19 generally is talking about in terms of cycles of the system
20 function, which may take a lot more than one operation per
21 cycle of system function.

22 MR. LEWIS: I was only saying one has to be
23 reasonably precise about what one means. I'm fond of
24 pointing out to people who speak of reliabilities of 10 to
25 the minus 6 per year that the history of the human race

1 shows that the laws of nature change a lot faster than that.
2 Anything we say about mathematics, for example. It was you,
3 Rich, who mentioned Euclid. Euclid held out for about a few
4 millennia before people discovered that space didn't need to
5 be flat. Then alternatives showed up. So we have to be
6 very careful about these extraordinarily low probabilities
7 we discuss, because we discuss them in a limited context
8 that may not stand the test of time.

9 MR. CATTON: And if I leave my screen saver on it
10 screws that up; it locks it up if I leave it on for several
11 hours.

12 MR. LEWIS: I always turn my screen off when I
13 leave and leave the computer churning away.

14 MR. KEMMERER: That is a better screen saver.

15 MR. LEWIS: It's an excellent screen saver, as a
16 matter of fact.

17 Dick.

18 MR. KEMMERER: I got a little confused with what
19 Pat said and then what Richard said. As I remember the
20 Patriot, it was having a drift that had to be reset within
21 19 hours. It seems Richard then said, well, using the four
22 channels and resetting one of them would take care of that.
23 I know nothing about your voting algorithm, but it seems to
24 me that if you reset one of them, that's the one that you
25 would probably vote out at the next round that you went to,

1 because it now has the correct result but it's different
2 from the other three. I don't know if you vote part of
3 these four out or you are handling that. Maybe you'd have a
4 premature shutdown.

5 I'm a little confused about the solution to the
6 problem.

7 MR. PLACE: I raised the Patriot because it was an
8 example of drift where a reset could fix the problem. They
9 had a simple system. It wasn't voting in conjunction with a
10 bunch of other like minded pieces.

11 If you do take the reset view, then you need to
12 again complicate something else, which is your voting
13 algorithm. You need to be able to say, well, it's out of
14 synch right now but it's coming back into synch in terms of
15 the rest of our system.

16 My understanding is that voting for power systems
17 is not one of four you trip; it's two out of four that
18 causes a trip. So even though it's out of synchronization
19 with the other three at the moment, it's not going to
20 guarantee a trip at that time.

21 MR. KEMMERER: As I said, I'm ignorant of all
22 that.

23 MR. PLACE: There are certainly dangers with that
24 approach and it's one that needs investigation. It seems
25 that there is no easy answer. We're going to shift the

1 problem from one place to the next and it's a case where we
2 want to shift it to. If we determine that memory systems is
3 a big problem for analysis, then let's start thinking about
4 how we can reduce the amount of memcry that these systems
5 have. Reset is one possibility. Maybe there is a while
6 when you can't trust that channel that you rest until it has
7 come back into synch. I'm not claiming that I have the
8 answers. Just possible suggestions.

9 MR. KEMMERER: Actually we got here because you
10 asked us what we thought NRC should do, whether they should
11 use a formal approach in their development. I don't know
12 how many nines you'll get or how all this fits in, but it's
13 my belief that whenever you are developing any software
14 where it's critical that it's correct that you should use
15 formal methods on it. Whether getting the software right
16 still makes the system wrong or not, it's no excuse for not
17 getting the software right, not getting your specs right.

18 MR. SHEWMON: You said one should use formal
19 methods. Probably everybody around the table would agree
20 with that. I'm a little intrigued that the other Richard
21 down at the end didn't like the procedures that the IEEE
22 puts out, and ASTM, because those are craft. To what extent
23 is there a set of generally approved formal procedures? Or
24 is one's formal procedure somebody else's craft?

25 MR. KEMMERER: Actually I have that down as

1 something I wanted to comment on. I don't particular read
2 standards. I find them terribly boring. On the other hand,
3 I believe that the software development standards are
4 something that you should adhere to but using the
5 formalizing with them. You said that you could use your
6 approach with those standards. I think that's what you
7 should push for rather than saying that those standards
8 aren't good. The standards aren't enough. You did say
9 that, but after saying don't use these standards.

10 MR. SHEWMON: I would like to know to what extent
11 is there a generally accepted set of standards for formal
12 methods so everybody knows what formal methods means.

13 MR. KEMMERER: No.

14 MR. COBB: That does not exist today.

15 MR. KRESS: When you say formal methods are you
16 talking about mapping the input space onto an output space
17 that is known and comparing the known to what you get? I'm
18 not quite sure what you mean by formal methods.

19 MR. KEMMERER: Let me throw the ball back in your
20 court. If you give me a specification for a function that
21 you have now, which is probably written in English, then
22 what I would proceed to do is to represent that formally
23 mathematically. For instance, going to the trivial, if you
24 say you have a sort routine where you want to sort all of
25 the values of an array and some range 1 to n in ascending

1 order, what I would do is proceed to write something that
2 says for all index "i" in the range 1 to n, a sub i is
3 greater than a sub i minus 1. I can't get it exactly right
4 here without writing it all out for you, but something of
5 that sort.

6 MR. KRESS: That's a formal method of translating
7 the specifications. I was assuming we meant a formal method
8 of assuring that what you did actually did what you thought
9 it was going to.

10 MR. KEMMERER: Are you talking about the
11 implementation?

12 MR. KRESS: Yes. After you put it together, does
13 it actually do that under all circumstances?

14 MR. KEMMERER: I'm sorry I didn't bring my
15 tutorial slides with me. I'd love to jump up and do this
16 right now.

17 There are a number of levels. First off you want
18 to have a formal specification for the system. This is what
19 Richard was talking about. Then you want to refine that
20 into formal specifications for your components. Again, I'm
21 just reiterating what he said.

22 There are several things you can do. At the
23 system level you can test your specifications, which is
24 something that I push. This is a case where if we had our
25 peers, our designers or the customer, and so forth, where

1 you could ask a question about, okay, I present the formal
2 specification to you and you have a scenario, well, what
3 would happen if. You mentioned before the temperature is
4 above this level, and whatever, and what is the result.
5 With a formal specification, because it's precise, I can put
6 that in as my current state, execute my specifications and
7 say here's the result you would get. It is important to
8 test your spec before you actually get down into the refined
9 design and refined specifications, and so forth.

10 Once you are down to the code level -- I'm taking
11 some quantum jumps now -- I'm down to where I have a
12 specification for the code of a particular module or a
13 particular procedure, I have an entry and exit
14 specification. Entry says what am I assuming about the
15 inputs to this module. I may be assuming nothing at all, so
16 I have an entry assertion of true. Or I may be assuming
17 it's in some range. If I'm assuming the variables when I'm
18 making a call to this procedure satisfy something, in
19 everyplace where I call it I have to guarantee that it
20 satisfies that entry assertion.

21 Once I have this entry assertion, for every
22 statement in the program there is an axiomatic definition of
23 what it does, what's the effect of that program.

24 I could show you some examples of that but I have
25 trouble saying them right here.

1 So what you do is actually execute that program.
2 There are several approaches. One is you start with the
3 exit assertion and push it backwards through the program,
4 which is known as the weakest precondition. When you get
5 back to the entry assertion you show the entry assertion is
6 strong enough to imply whatever that weakest precondition
7 was.

8 Another approach is the forward direction symbolic
9 execution where you start off with your entry assertion, go
10 through each step. When you get to the exit you show that
11 the result is strong enough to imply the exit assertion that
12 you want.

13 This is all done mathematically. At the code
14 level it requires formal semantics for the language.

15 MR. SHEWMON: How many languages meet this formal
16 requirement that you just mentioned?

17 MR. KEMMERER: None of them for the complete
18 language.

19 MR. COBB: I agree with everything that was just
20 said. I think the simple definition of a formal approach to
21 software is to regard the specification for the software as
22 a definition of a function and then the refinement of that
23 function into executable code. You do it in such a way that
24 you have developed a formal proof that the rule that you
25 have developed provides exactly the same behavior as the

1 initial function that you have. That means as a
2 precondition to that that you have to have the function
3 defined in a way that is mathematically viable.

4 One thing I did say was that the function that you
5 define to begin with for the system or the module as a whole
6 ought to be a black box function so that you have no
7 internal inventions in that function. That's a very
8 important ingredient that I think has made the so-called
9 formal methods much more useful and friendly to
10 mathematically competent people for reasoning.

11 The second thing we can say about formal methods
12 is that in the process of going from a function to a rule
13 there are basically two classes of formal methods.

14 There is a class of formal methods that is called
15 axiomatic verification where you have preconditions and
16 post-conditions and you do reasoning like was just
17 explained.

18 The other class of formal methods that has been
19 developed is something that is called function verification.
20 Every part of a software system has a function in and of
21 itself: an assignment statement has a function; an IF THEN
22 ELSE statement has a function.

23 For any function to rule at most three things have
24 to be shown. Proof arguments are finite. You can get valid
25 proof arguments that are easy to do. And you know what you

1 have to show for every different kind of structured
2 construct for your language.

3 For your question about what kind of language, if
4 you use a functional verification type of approach, you
5 limit your language constructs that you use in your rule to
6 only the language constructs that are defined by a
7 structured program: a DO ODD, which is sequence; and IF THEN
8 ELSE; a DO UNTIL. If you limit only those things to those
9 kind of language constructs, we know what to show for every
10 kind of those constructs, so we know how to do it. Where
11 you get yourself into trouble is where you use these kind of
12 control flow things like GOTO, or these interrupt driven
13 constructs and all of these things that extend these
14 programming language to very complex things. They may
15 introduce some efficiency in your program, but they
16 introduce constructs that you can't say anything about.

17 The crucial ingredient in making proofs is not so
18 much the control flow but is the data structures that you
19 are manipulating, whether you are manipulating a sequence,
20 an array, or all of these different mathematical structures.
21 Doing proof assertions on the data is more complex than on
22 the control flow, and that is one of the reasons that in
23 these kind of programs you want to use simple data
24 structures.

25 From the standpoint of what is a formal method,

1 there isn't a set of standards for these things. I think
2 there is a common agreement among the people who are doing
3 them. I may not use the same style of proof that somebody
4 else is, but people would recognize my style of proof and I
5 would recognize their style of proof. People have different
6 styles of proofs. The important ingredient is that you are
7 doing a proof, that you are not just hoping. There's a big
8 difference between hope and proof.

9 If we are infallible, proofs would always be
10 right. The difficulty is there have been some theorems that
11 have been published that turn out to be wrong. That's what
12 we've got to minimize against by all these other kinds of
13 controls that we are putting in.

14 MR. LEWIS: There is always a certain probability
15 that a theorem that has been proved will be wrong.

16 MR. SHEWMON: One of the things the NRC gets
17 criticized for a lot is, well, I'll know one when I see one;
18 I can't tell you what it is, but bring me one and I'll
19 recognize it. What you said sounded close to that to the
20 inexperienced ear.

21 MR. KEMMERER: Actually, I think I would worry
22 more if you had the standard for it. I don't think the
23 problem is with the standard. The standard is going to say
24 that you're going to be using some form of first order
25 predicate calculus to express your specification and you are

1 going to do this. This turns out to be the process. The
2 process with the wrong specification isn't going to buy you
3 anything at all. One of the things Rich said earlier is
4 that when we start off with a spec we have no way of proving
5 that the spec is right. That's where we start off. That's
6 our requirements for the system, the specification. In the
7 formal method that's the first time we have formalism.

8 When I teach formal specification verification
9 classes at the university I always give an assignment to
10 write a spec for a system, a library, database or something
11 of that level. I know the mistakes the students are going
12 to make, but by the process it looks right. It's formalism.
13 They've got their FOR ALLs and their EXISTSs and so forth,
14 but in this case they often have inconsistencies, as Hal
15 said.

16 You have to be careful about the idea that if I
17 have a standard that says here is the process you are going
18 to take that you think you're going to get a better product.
19 There's a big difference between process and product.

20 I think anybody who has built software and worked
21 on large systems, you know that there are people you'd like
22 to get working for you. When Harlan did the original
23 cleanroom work at Federal Systems that group of people that
24 he had doing that was one helluva good group of people. The
25 same way when IBM did the Chief Programmer technique. The

1 first time they did that for The New York Times database
2 they had the best compiler person, the had the best database
3 person. It was doomed to success.

4 So you have to be careful. It's nice to have a
5 process. It's nice to have this checklist to say that I've
6 done all of this, but you've got to be real careful not to
7 say that because I've followed this process I've got a good
8 product.

9 Now I'd like to tell you what the solution is now
10 that I've told you what the problem is, but I don't have it.

11 MR. LEWIS: In fact the last thing you said is
12 precisely why I don't ever read IEEE standards. They always
13 define the process but they never say much more than that.

14 MR. PLACE: The process is designed is designed to
15 make sure that the implementation conforms with the
16 specification. As is being stated quite clearly, we have
17 two problems. One is, what is it we are going to build, and
18 the second is, did we build what we wanted to build. If by
19 the process we can eliminate that second problem, did we
20 build what we said we wanted to build, we can concentrate
21 our efforts very much on the first problem, which is, are we
22 building the right thing at all.

23 I think the process does have some benefit.

24 MR. LEWIS: The availability of the language
25 doesn't make it possible for me to write Shakespearian

1 sonnets.

2 MR. PLACE: But the absence of the language
3 doesn't deter you from writing them.

4 MR. LEWIS: But I know all the words in them, as a
5 matter of fact.

6 I have a feeling that some of us have been
7 dominating the conversation. There is a statistical term
8 known as extending the conversation which is used in proving
9 theorems.

10 Let me first ask some of the more quiet members of
11 the Committee whether they want to intervene.

12 MR. WYLIE: Let me ask one. This goes back to the
13 discussion of common mode failures. What is your opinion of
14 the feasibility and practicality of designing hardware and
15 software that you can specify, design and implement for the
16 reactor protection system and safety systems such that the
17 probability of common mode failure is negligible?

18 MR. LEWIS: Who wants to take that one?

19 MR. WYLIE: From the discussion I've heard here
20 I'm more skeptical now than I was.

21 MR. LEWIS: You are being overwhelmed by silence,
22 Charlie.

23 MR. COBB: I'll say something about that. In
24 order to worry about the issue of common mode failure I
25 think one has to identify at the specification level each of

1 the potential sources of having a common mode situation
2 occur. Once one has that done, then one has to figure out a
3 way of mitigating against each of those possible sources of
4 common mode failures.

5 The question that you are asking is what is the
6 possibility that I will be able to find a mitigating
7 solution for each possible source of common mode failure. I
8 would suspect that at the specification level we ought to be
9 able to find that solution. Specifically in the province of
10 this committee, which is talking about software type of
11 issues. What is the probability of being able to eliminate
12 common mode failure in your software items that are part of
13 your safety protection system where the same logic failure
14 in the implementation of the software is going to cause a
15 failure of the safety system at the same time, which is the
16 software implementation part.

17 There are two solutions to that. One, of course,
18 is to have software that doesn't have a failure, assuming
19 perfect operation. By using these methods that we have been
20 talking about on this side of the table one has a very high
21 probability of having high quality components. So one is
22 mitigating the chances of a problem by having a high quality
23 thing.

24 The second situation is that you can have diverse
25 components. Even given the same specification, I can

1 certainly implement a different rule. What I'm doing in
2 that particular case is adding a cost to get diversity.

3 I'm not sure whether I would recommend that a
4 manufacturer do that or not do that. I think that's an
5 issue that has to be thought about. But the point is that
6 given that software is a mathematical object as opposed to
7 being a physical object, I can in fact create diversity if I
8 desire to create diversity. I can create diversity at that
9 level. So I can really deal with common mode failure at the
10 software items. I'm not sure know how to deal with common
11 mode failure with power shortages and these external events,
12 but that's independent of whether I have software or don't
13 have software in this channel.

14 I think the presence of software in your safety
15 system can enhance your protection against common mode
16 failure because you can build better logic into your
17 decision analysis components that recognize that I may have
18 some common mode failures happening out there in the reactor
19 and the sensors.

20 One of the things that people have mentioned to me
21 as a source of a common mode failure is a sensor gets stuck
22 at some value out there in the reactor vessel. I certainly
23 can do something about that in software where I can't do
24 something about that if I have hardware.

25 So there is lots of protection I can build in in

1 software to eliminate sources of common mode failure that I
2 couldn't otherwise.

3 My feeling is that for all sources of common mode
4 failure that people can identify you ought to be able to
5 build a defense against those and mitigate those For
6 sources of common mode failure that nobody has identified
7 you may not be able to build a defense against those because
8 you haven't yet identified them.

9 MR. WYLIE: Those are the ones that bother you.

10 MR. COBB: I think everybody over here has made a
11 real focus that the specification is the crucial issue, not
12 the implementation.

13 MR. WYLIE: I agree.

14 Would this suggest that there should be separate
15 groups of people assembled to attack these and develop the
16 hardware and software? Since they all think alike, is it
17 possible to put the common mode failure in by the design?

18 MR. PLACE: I'm not convinced that just by
19 separation you'll get diversity. I think you need to be
20 active about being diverse. There needs to be communication
21 between the groups so that they are actively attempting to
22 be different from each other. If you throw two groups into
23 solving some problem, especially when they are down to
24 relatively small component level problems, you end up
25 running the risk that they will both make the same

1 interpretations or misinterpretations and interpret things
2 in the same way, and in fact you end up with very similar
3 pieces of software that will suffer from the same problems.

4 The second thing I want to address with respect to
5 diversity is that if you have diversity by components and
6 think you've solved your problem of common mode failure, I
7 think you lead yourself into an erroneous conclusion,
8 because there's the problem of between the system and that
9 set of components there was something we call design, and
10 that was the splitting up of the system into that particular
11 set of components. That can lead you into a common mode
12 failure if you combination of components has some flaw
13 within it. It doesn't matter that each of the components is
14 diverse; the system as a whole still has that problem.

15 There is still a problem and we have to be very
16 careful what we mean by diversity or where diversity goes.
17 It has to start from the very start with the specification.
18 A second specification might be the safer way to go for
19 diversity.

20 MR. COBB: I agree with that statement. That's
21 the reason that in that little set of documents that I had
22 for the specification I had a separate document at that
23 level of analysis of the decomposition to analyze that
24 particular question all by itself. That was the document
25 that I called safety analysis. That's a very important

1 thing that has to be analyzed.

2 MR. WYLIE: All of you gentlemen have had
3 interaction with the Department of Defense, I assume.

4 MR. KEMMERER: I have.

5 MR. WYLIE: Then you have some experience with
6 what they have done in development of these systems. They
7 face the same problems of common mode failure and things
8 like th's. Is it your opinion that it is practical and
9 feasible to design a complete digital computer type system
10 for these reactor protection and safeguard systems so that
11 common mode failure then is negligible?

12 MR. LEWIS: It's interesting that DoD makes
13 everybody write in Ada which is not a language which is
14 distinguished for a few simple data constructs and a few
15 simple control constructs. I'm just adding to the question.

16 MR. KEMMERER: I agree with what you just said and
17 there are lots of things in the literature from people who
18 you might call formal methodists that nail Ada for exactly
19 that reason.

20 You seem to be asking one question and we may be
21 answering another one. I think your last question was --
22 and I don't know what DoD had to do with it --

23 MR. WYLIE: I referenced DoD because they have
24 these systems guiding missiles and other things.

25 MR. KEMMERER: I didn't mean to mislead you, but

1 the DoD that I deal with was in terms of security where we
2 were proving systems were secure. Not safety. I've never
3 done a proof that anything wouldn't cause cancer to
4 somebody. It might be secure but it still might cause
5 cancer.

6 The question you asked was, can you build a
7 digital system that will guarantee negligible chance of
8 common mode failure? I guess I want to ask you, can you
9 build an analog system that will guarantee negligible chance
10 of common mode failure?

11 MR. WYLIE: It has been accepted. Let's put it
12 that way.

13 MR. KEMMERER: The thing to look at here is we are
14 replacing a component, possibly a hardware component with
15 software here, and you should have your specifications for
16 that hardware component and we're proposing that when you do
17 your do your software do the best job possible on that
18 software. What we are promoting here is formal methods.

19 Your common mode analysis, I would assume, on the
20 hardware system was done separate from the implementation of
21 the hardware. I'm not sure. I think that same kind of
22 analysis has to go on here. I don't believe you're going to
23 get it by design diversity, and I think that Leveson and
24 Knight showed problems with that design diversity. They
25 reran experiments that were done by Kelly where they tried

1 to say, well, what you do in the hardware you can do in the
2 software.

3 The thing we have been stressing here is getting
4 the specs right. We also pointed out that the place where
5 you are most likely to get a failure is because you haven't
6 considered some special case. One of the things that
7 Leveson and Knight showed was that those special cases are
8 overlooked. When you take two teams to do it they both are
9 overlooking it.

10 You asked a question earlier about whether it
11 would be better to separate our teams and have one design
12 hardware and one design software. I think maybe it would be
13 better if they worked together and they both tried to do the
14 job right rather than taking what's a limited resources of
15 good people and get that high level spec right.

16 I didn't answer your question. I'm not sure what
17 ten nines or eight nines or six or however many Hal wants to
18 throw at me mean.

19 MR. SHEWMON: But he thinks it can be shown that
20 it's better than what's in second place is what I heard.

21 MR. KRESS: The common mode failure question is
22 more a hardware question than a software one anyway. The
23 question is, are digital systems which includes software and
24 hardware more vulnerable to conditions that would cause a
25 common mode failure? These conditions might be shaking due

1 to a seismic event or pressures and temperatures and
2 humidities due to upset conditions. Those are hardware
3 questions more than software.

4 I think your question was, is there something
5 unique about the fact that you have software as part of this
6 process that would make them have different vulnerabilities?
7 Frankly, I don't think there are.

8 MR. KEMMERER: Because you are putting a digital
9 system, the things that are different are going to be the
10 hardware things, which is the question that I thought Paul
11 asked this morning. Then when I asked him later whether it
12 was answered, he said, no, he asked a different question.

13 MR. SHEWMON: So tell me the question I should
14 have asked.

15 MR. KEMMERER: One of the assumptions we are
16 making about formal methods is that the hardware performs as
17 it's supposed to. If you get a sunspot or a shake or
18 something and a bit rolls the wrong way, you can do all
19 those proofs that your program is going to perform as it's
20 supposed to, but those proofs are based on the fact that the
21 compiler is implemented correctly as per the semantics of
22 the language, which is something that Hal brought up
23 earlier. The other thing is that the hardware performs as
24 it's supposed to.

25 MR. CATTON: Don't you have to do a final test of

1 the embedded system, the hardware and the software?

2 MR. KEMMERER: Yes. But what's that test going to
3 tell you? You've tested your embedded system and it says it
4 works okay. You go over and you shake it and says it worked
5 okay when I shook it, but what if Hal shakes it different
6 than I did?

7 MR. LEWIS: What if I were left handed?

8 MR. KEMMERER: Right.

9 MR. CATTON: You sort of know what happens to
10 hardware. Don't you have to input what the hardware could
11 give to the program even if it's not quite what you would
12 expect under normal circumstances?

13 MR. COBB: I agree. The domain of usage in these
14 particular systems has to take into account the changes in
15 stimulus as a result of the hardware. That's one of the
16 reasons that they have this voting and things like that.

17 MR. CATTON: I thought that's what you were
18 talking about when you said a Markov type input, that you
19 would have some sort of a distribution of expected inputs
20 from a piece of hardware and you'd have to do a number on
21 it, and the distribution would include probability of off
22 norm.

23 MR. KEMMERER: Those are two different issues. I
24 think we are confusing two different issues. One issue says
25 what inputs do you get to the software where that input is

1 generated by a piece of hardware. The other thing says that
2 the software is running on some platform, and if that
3 platform perform as it's supposed to, that software is not
4 going to perform as it's supposed to. I'm talking about the
5 processor that the software is running on. Those are two
6 different issues.

7 The formal proof isn't going to prove anything if
8 the underlying machine that is executing those instructions
9 isn't doing what it's supposed to.

10 You can push that stuff down. You can do proof of
11 compilers; you can do proof of the operating system; you can
12 proof of the hardware. Eventually you are going to get down
13 to solid state physics and we'll throw the proof over to you
14 guys.

15 MR. CATTON: Basically what I'm talking about more
16 is the specification. You're specification had better
17 include possible off norm behavior of the hardware.

18 MR. KEMMERER: Off norm behavior of sensors and so
19 forth. Absolutely. That should be included as part of your
20 specification and then you can do the proofs about that.
21 But if the processor that the software is running on doesn't
22 do what it's supposed to, then all bets are off.

23 MR. CATTON: I understand.

24 MR. COBB: Except I think that that's the reason
25 the designers of these systems have got four parallel

1 channels and they are doing the voting. They are trying to
2 take into account these possibilities of hardware failures
3 and they have these kind of switches so they can know
4 whether the hardware has failed and things like that. There
5 are a lot of things that can be done to do that. That's why
6 at the system design level you have to deal with the kind of
7 issues that are just being brought up here, and the
8 component design issues you have to deal with assuming that
9 the processor is going to work correctly.

10 There are a lot of levels of concern. In doing
11 the analysis you have to separate the levels of concern.

12 MR. CATTON: But not forget any of them.

13 MR. COBB: That's an important issue, yes.

14 MR. KRESS: Is there is a way for the system to
15 continuously check itself to say, yes, the hardware is
16 implementing my software correctly because I know it should
17 do this and that and it makes regular checks?

18 MR. COBB: There is certainly a way to specify
19 that in the specifications. I think some of that should be
20 built into the specifications. On the other hand, I think
21 if you build in too many checks, then you increase the
22 complexity of the software and complexity of the system.
23 One of the challenges of the designers of these systems is
24 to find the right tradeoff, which is the same challenge all
25 engineers have, finding the right tradeoff. We're talking

1 about the probability now of a 386 chip failing versus the
2 probability of a relay getting stuck. At least my limited
3 experience with a 386 chip versus relays is that I'm all in
4 favor of 386 chips.

5 MR. LEWIS: There's no contest.

6 MR. COBB: My relays don't work as well as my 386
7 chips. I can guarantee you that.

8 One other thing I wanted to mention is that from
9 the standpoint of verifying these specifications, a
10 specification is a model of desired behavior. In terms of
11 what I have suggested, a black box function is one model of
12 desired behavior of one of these components. The Markov
13 model is another model of anticipated behavior.

14 These two models and viewpoints are complementary.
15 One is sort of taking one kind of view and the other is
16 taking another view of the system. One of the things that
17 we are discovering in helping get specifications right is
18 that by the time you get done these two models have to be in
19 complete conformance with each other, in complete agreement
20 each other. One of the things that you are getting by
21 having two different models and views of behavior is that it
22 really does help you get to the right specification. We are
23 beginning to find that that's a very valuable help to take
24 two different views of the same thing and formally make them
25 come together. That has been a big help to us.

1 MR. LEWIS: One of the responsibilities of the
2 Chairman is to reward the people who have been patient and
3 have had questions bubbling out from them but have been too
4 kind to push. At least one member of our audience has been
5 waiting for two hours to say something, and by golly, it's
6 your turn. Step to a microphone and tell everyone who you
7 are.

8 MR. MILLER: My name is Lance Miller. I work for
9 Science Application International Corporation. We have a
10 contract with NRC and Electric Power Research Institute to
11 develop guidelines for the verification and validation of
12 expert systems in nuclear applications and that's where I'm
13 speaking from.

14 I appreciate the opportunity to act as possible a
15 devil's advocate here, although I am a believer in
16 mathematical methods, formal methods. But there is an
17 implication in the necessary simplification of the
18 presentation of sufficiency. That's the nature of my
19 comments.

20 With respect to formal verification, there was the
21 quasi-assertion that it can handle all of the
22 characterization that you want, that if it's given a formal
23 specification of the requirement you could test for it
24 either in the design or in the implementation. It's very
25 difficult, for example, on the other hand to test for

1 safety. How do you formalize safety? You can formalize a
2 function that's supposed to be achieved, but it's very
3 difficult to formalize safety except by pulling on that
4 thread and then finding all the ways in a particular
5 instantiation that there is non-safety. That requires a
6 formal representation language for the requirements through
7 the design, through the implementation. There are many
8 issues that haven't been addressed there.

9 Having worked with Harlan for several years
10 adjacent to him in similar tasks, I'm very much aware that
11 the cleanroom process is a human process. There's a great
12 deal of training going on there. I would just point out
13 that all human processes are variable. You can't just say,
14 well, let's do it this way. There is a lot of training.
15 Chief programmer is the best, and how do you get that? It's
16 a complex process.

17 Even in DOE's Lawrence Livermore lab where they
18 use a formal method called FAM, when you actually get down
19 to looking at that particular procedure, it's guided very
20 much by the intuitions of the person who is using it, and
21 they aren't accessible to the rest of us. Having used
22 Prolog a lot, I'm very familiar with first order predicate
23 and with its deficiencies. It's very limited.

24 In the area of NRC regulation you want to be able
25 to express failure concepts and safety concepts. Really

1 what you would like is modal logic, larger logics that can
2 handle possible worlds and things can or cannot occur. As
3 people know, there has not been an acceptance of modal
4 logic. Really we're back down to first order. First order
5 doesn't handle a lot of concepts very well.

6 It doesn't handle time very well. If you had a
7 requirement that there should be no hazards, no temporal
8 conflicts, everything should be just in time, and so on,
9 it's very difficult to express these concepts formally.
10 What you will find, although it has been solved a number of
11 times in the AAI literature how to do this, people will vary
12 enormously in their style of doing it. Where one persons
13 style is adequate or not is a question.

14 I think we have to be very concerned whether it's
15 going to be the end-all. I don't think that was the
16 implication, but it is a simplification here and there is an
17 emphasis from our point of view in reviewing what should be
18 guidelines for expert systems being embedded. We find that
19 expert systems have only one component which is really quite
20 different from the other components of conventional
21 software, and that's the knowledge base.

22 We think of our job as very much a conventional
23 software job and limited in certain respects. It's our
24 opinion at this time -- this is not the NRC's opinion; this
25 is the contractor's opinion -- that no single method is

1 going to be sufficient, that you have to have a multiplicity
2 of means and you have to look at the full development cycle.

3 Starting with requirements is clearly an essential
4 aspect. The suggestion here was that both for the
5 requirements specification and the design specification that
6 peer review processes were sufficient. We would feel that
7 that is not the case at all. There must be use of so called
8 "upper case" tools, automated computer tools to verify
9 things such as the absence of conflict or inconsistency.
10 Humans just simply can't do it in a peer review process.

11 The requirements tracing process is essential to
12 determining the presence of all implemented functions and
13 the absence of unintended functions. Talk about craft.
14 That's a very arcane capability. There are some tools
15 around, but it's essential to the process. If you fail to
16 do that, you won't be able to guarantee what you've done.

17 There are two specs that are important, the
18 requirements spec, the articulation of all the requirements
19 and their detail, and the design spec. Too often the
20 clarify of the requirement spec is not addressed.

21 For example, in a recent procurement that we
22 competed in at the FBI for the National Crime Information
23 Center the requirements spec was maybe two and a half fee
24 high of documents that went into enormous detail about the
25 requirements for the various interfaces. Characterizing

1 these in a formal way or in some way and then animating
2 these is essential. These two comments apply also to
3 design.

4 One thing that I think is continually overlooked
5 in the development of highly complex interactive systems,
6 particularly for DoD, is the concept of operations, the ops
7 concept. For example, in the GPS system that IBM had a lot
8 of involvement in, and I was involved to some extent. The
9 concept of operation was delayed to a very large time. It
10 was found after 85 percent of the funds were expended that
11 it was a deficient concept. It didn't allow for a number of
12 anomalous situations that had the concept been explored it
13 would have revealed.

14 The shutdown software that we are talking about
15 here as well as most of the other software will involve
16 complex interfaces both with the data channels and the user.
17 Therefore exploration of the con ops is an extremely
18 important aspect. It's almost entirely underrated or
19 undervalued in reviews. So I suggest that to you for
20 consideration.

21 The last point is with respect to testing.
22 Testing was presented as a way not to discover defects but
23 rather as demonstration of the correctness. We feel because
24 of the problems of verification that that is a limited view
25 perhaps. There should be a variety of types of testing.

1 Structural testing, as everybody is certainly coming to
2 agree on, is probably the least important. But there are a
3 number of other testings. You have to do functional
4 testing. One of the things that's typically not done in
5 functional testing is to try to devise tests that will cause
6 the function not to occur. That's really underrated.

7 Another area is domain testing where you are
8 looking at partitions and domains and you are really at the
9 crossover points and you really want to select. Your sample
10 is such that it's right close to the boundary but if it were
11 a little bit more it would cross over into a boundary of
12 some other area. This is an arcane area but it's extremely
13 important and it's not sufficiently done.

14 We would also recommend very highly failure modes
15 effects causality analysis, trying to determine ahead of
16 time potential safety or hazard flaws, and then trying with
17 the software developers to determine where in the code these
18 could occur and then testing for them.

19 A major category of testing would be robustness
20 testing. This is an analog to the accelerated live testing
21 that we do with hardware. The notion is to try to identify
22 all of the design parameters and assumptions with respect to
23 any particular module or aspect of the system and then to
24 systematically violate them, corresponding to handing a toy
25 to a child and letting him bang on it and shake it and so

1 on. This can be done systematically.

2 There are other issues. There are larger issues
3 such as process. How do you do your testing? An assertion
4 is you ought to do your testing such that the next
5 expenditure of a testing dollar addresses the most serious
6 remaining fault. Then you ought to repair it, and your
7 repair strategy ought to be carefully designed and
8 implemented.

9 One final thing. In terms of the failure model,
10 systems can fail not because they are incorrect because they
11 are used in a different context than intended. The Patriot
12 is a good example. It was intended to protect a few clicks
13 in circumference, never a whole city. All of the
14 assumptions about the ballistics of the incoming missile and
15 so on were for that, and the 40 percent rate that we had is
16 because it was a perfectly good system applied to a
17 different problem. So that's another context that you have
18 to make sure that the software you are using is really in
19 the context for which it's applied.

20 Finally, I think in general we need the process
21 issues. I think continuous process improvement is essential
22 to achieving the certainty of high reliability and therefore
23 that backs into a life cycle. You would say, what should be
24 the life cycle. So I want to do my process improvement
25 within a particular project, and that would be incremental

1 system builds. Then what you do when you find an error is
2 you say why did that error occur, what process is it due to,
3 and then you improve that process.

4 Thanks very much.

5 MR. LEWIS: Thank you.

6 I think the time has come to declare a lunch
7 break. Why don't we reconvene in 62 minutes. I'm rounding
8 everything to the next big number on the clock.

9 [Whereupon at 12:03 p.m. the meeting was recessed
10 to reconvene at 1:05 p.m. this same day.]

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

AFTERNOON SESSION

[1:00 p.m.]

MR. LEWIS: Let's reconvene the meeting of the Advisory Committee on Reactor Safeguards, Subcommittee on Computers in Nuclear Power Plant Operations.

This morning we heard from some consultants and external advisors, who contributed a great deal. I thought it was a good session this morning. This afternoon we are going to devote to hearing what it's like out in the trenches, I guess, where it's all really happening. I guess Tom Pietrangelo from NUMARC will start the ball rolling and then we will have some people from the industry who have had experiences with AD conversion problems which is a specific item we have dealt with in the past. We are all yours.

MR. PIETRANGELO: Good afternoon. Thanks again, for the opportunity to address the Subcommittee today. It's always a pleasure to come and talk to the ACRS on different issues we come up with at NUMARC.

MR. CARROLL: Flunked another test, he lies.

[Laughter.]

MR. LEWIS: You took the words out of my mouth.

[Slides.]

MR. PIETRANGELO: First of all, I want to discuss a little bit about what NUMARC's role is on this particular issue, analog to digital replacements for I&C. NUMARC's role

1 is really a support role to Electric Power Research
2 Institute. EPRI has taken on an initiative and a leadership
3 role in the industry to facilitate the introduction of
4 modern digital technology into our power plants.

5 It's a broad scoped, far reaching plan that seeks
6 by the year 2000 to have a vision accomplished at several
7 pilot plants in the industry. NUMARC's role in support of
8 that initiative is to provide a unified point of contact for
9 the industry with the NRC. That's the extent of our
10 participation.

11 I will talk a little bit more later about a
12 Committee we formed with EPRI to try to address this issue.
13 Like I said, we were happy to participate also in last
14 month's full ACRS meeting that focused on 50.59 questions.
15 That was a very important discussion. The draft generic
16 letter that came out, we did offer some views at that time.

17 Equally important though, is what we are here to
18 talk about today besides the USQ question. That is a
19 defined stable process for licensing analog to digital
20 upgrades. I think we made the point last month that that is
21 equally important with the USQ question for both NUMARC and
22 the industry and licensees.

23 Regardless of whether a potential modification is
24 an unreviewed safety question or not, licensees still need
25 to know what the NRC's expectations are on what to do to

1 license one of these modifications. We need to know what
2 the requirements are, the comment objective being to
3 implement these modifications in a safe and reliable way.

4 I just wanted to make a few comments about the
5 discussion this morning. I found personally interesting the
6 discussion on the common mode failure question with the
7 software, and I think the Committee focused in on that much
8 the same way the industry has. We took the position in our
9 comments on the draft generic letter that was issued last
10 fall, that the potential for common mode failures in the
11 software is not a new phenomena; that it has always existed
12 whether it was hardware or software in redundant systems and
13 components.

14 I would encourage the Committee in your
15 correspondence to the Commission, to give your views on that
16 particular question because that is somewhat essential to
17 this determination of the USQ and the 50.59 process. That
18 has been singled out as the key issue in that arena. We are
19 very interested in hearing your views on that.

20 I would also like to give you a little status on
21 what our Committee is doing. We have met several times last
22 year and once this year, and have also had a few
23 interactions with the staff on a guidance document we are
24 developing. What we are after with this guidance document
25 is to establish this regulatory framework for licensing

1 digital I&C upgrades.

2 We have a meeting scheduled for tomorrow, and we
3 are hoping to issue the document for comment to both NRC and
4 our membership next month. It's coming along and the
5 process is continuing, and we expect to have a lot of
6 interaction with the staff and with our own membership on
7 that document.

8 That will give you a sense of what the document is
9 about at this point. We are really trying to establish a
10 road map in the licensing process referencing what written
11 guidance is out there and has been accepted already, getting
12 through the 50.59 process, and dealing with both the left
13 hand side of a chart we have developed which is when you are
14 still in the 50.59 and it's not a USQ and if you have
15 determined it's an unreviewed safety question, establishing
16 the guidance for what to do at that point.

17 We are looking at both sides of that question,
18 again, the intent being to stabilize and define the
19 regulatory process for these modifications.

20 The last thing I would like to mention is NUMARC
21 made no effort to coordinate the utility presentations that
22 you are going to hear this afternoon. There is no
23 predetermined message we are trying to send the Committee in
24 these presentations. This is the utilities' experience with
25 some digital modifications that they have had over the last

1 couple of years, and that experience speaks for itself.

2 Without further comments or questions, we can get
3 into the utility presentations.

4 MR. LEWIS: I have a terrible memory, but I
5 thought we had written a letter on AD version; am I
6 wrong? Maybe it was just a nice dream - nad.

7 MR. PIETRANGELO: You did, in fact, write a
8 letter. I think what I was getting at was this potential
9 for common mode failure in the software question.

10 MR. LEWIS: I understand. Maybe I drafted one and
11 it didn't pass.

12 MR. CATTON: You did grumble a lot about it during
13 the letter writing.

14 MR. LEWIS: Yes.

15 MR. CATTON: Maybe somebody must have taken it
16 out.

17 MR. LEWIS: Somebody may have. That may be what
18 happened.

19 MR. PIETRANGELO: My recollection is that you
20 didn't address that particular question in the letter.

21 MR. LEWIS: I suspect the Committee took it out of
22 the letter. That happens occasionally. Very good. Thank
23 you very much, Tony. Are there any other questions for
24 Tony?

25 [No response.]

1 MR. LEWIS: We will just go on to industry
2 experience. According to my agenda our next talk is from
3 Northeast Utilities, which I assume means the furthest North
4 or something.

5 [Laughter.]

6 [Slides.]

7 MR. VAN NOOREDENN: Good afternoon. My name is
8 Jerry Van Nooredenn, and I am the licensing supervisor for
9 Connecticut Yankee and Millstone Unit 3. We appreciate the
10 invitation we received from the ACRS through Stuart Long, to
11 provide a presentation today on Connecticut Yankee's
12 experiences with I&C upgrades. In particular, we are going
13 to discuss the upgrades that we have implemented on the
14 reactor protection system and the aux feedwater system.

15 In attendance with me today I have a large group
16 of people in the audience. I just want to recognize them.
17 They will be able to assist the ACRS in answering any
18 questions you may have when we get into the discussion
19 section.

20 I have Mike Bain here, who is the Engineering
21 Manager at Connecticut Yankee. Mike Brothers, the Systems
22 Supervisor at Connecticut Yankee. He will also be
23 presenting part of the presentation today. Tom Cleary, who
24 is a Licensing Engineer in my department. Joe Fougere, who
25 is the Project Engineer for the Feedwater system upgrade

1 that we are going to be implementing in May at Connecticut
2 Yankee. John Leger, who is the I&C Engineer at CY. George
3 Pitman, who is our Manager of Project Services for
4 Connecticut Yankee.

5 MR. LEWIS: That's quite a delegation. Is the
6 weather bad in New England?

7 [Laughter.]

8 MR. VAN NOOREDENN: No, the weather is fine. We
9 have a pending submittal with the NRC staff, so we are very
10 interested in --

11 [Laughter.]

12 MR. VAN NOOREDENN: Our agenda today will just
13 cover a few topics. The introduction, I will handle. Mike
14 Brothers will talk about the need for the upgrades, the
15 50.59 process that we used when we performed these upgrades.
16 Then, I will just have a few comments on the draft generic
17 letter and kind of a summary.

18 I just wanted to give you a little introduction on
19 Connecticut Yankee. Northeast Utilities have five nuclear
20 power plants, Connecticut Yankee being the oldest, Millstone
21 1, 2 and 3 and Seabrook, our newest. Even though
22 Connecticut Yankee is the oldest, it has the most modern
23 instrumentation and controls in it because of the updates
24 that we have been doing over the last five years.

25 Connecticut Yankee recently completed 25 years of

1 commercial operation. That was in January. We are still
2 going strong. We are very proud of this plant. Twice, it
3 set the world record for continuous operation. It was the
4 first light water reactor to achieve two runs over 400 days.

5 Presently, we have over 325 days of a continuous
6 operation. Our scheduled shutdown is in mid-May, so we
7 could easily have another 400 day run. We have excellent
8 SALP ratings. Our previous SALP period we had all SALP 1's.

9 Our first upgrade was completed over five years
10 ago. We broke up our upgrades into various phases, Phase I,
11 II and III. We did Phase I of the reactor protection system
12 upgrade in 1987, and then completed it in 1989. Mike
13 Brothers will get into some detail on that. In fact, he has
14 one of the microprocessor units right here that he can go
15 into some detail on.

16 In 1991 we changed the instrumentation and control
17 system on the aux feedwater system. We had run into a
18 problem. The old system was run by control air, and if
19 there was a sudden loss of control air we could get a
20 turbine overspeed trip. What we did is, we went from a
21 control air system to a hydraulic system that is now
22 controlled by computer.

23 This spring we are implementing Phase III of the
24 RPS upgrade. We are changing out the main feedwater system.
25 This may seem a little strange, in that Connecticut Yankee

1 is an older plant. The current system, there is not a
2 complete isolation between the RPS and the feedwater system,
3 so we are building that into our upgrade. That's why some
4 elements of the feedwater system also affect the reactor
5 protection system.

6 MR. LEWIS: It would be extremely helpful to me as
7 we go along if for at least one of these I could get some
8 idea of the complexity of what we are talking about, either
9 a circuit diagram or a block diagram or something
10 quantitative.

11 MR. VAN NOOREDENN: All right.

12 MR. LEWIS: That would be very useful.

13 MR. VAN NOOREDENN: I think Mike Brothers can go
14 into that. I might have something in my briefcase.

15 MR. LEWIS: That would be fine. We have a
16 tendency to talk in vacua.

17 MR. CARROLL: The feedwater upgrade is more than
18 just protection system though.

19 MR. VAN NOOREDENN: That's right. It's also the
20 control system.

21 MR. CARROLL: It's also the feedwater control
22 system.

23 MR. VAN NOOREDENN: Right. As Mike will get into,
24 one of the main reasons driving these upgrades is
25 obsolescence and problems with obtaining spare parts and

1 starting to have problems with reliability of the old
2 system. It's just worn out.

3 The final two points that I wanted to make in the
4 introduction is, so far after five years of operation we
5 have had excellent reliability, excellent experience with
6 the reactor protection system upgrade. We have not had any
7 scrams or software failures, or anything of that nature, in
8 five years of operation.

9 The aux feedwater system was a commercial
10 dedication, which we completed successfully. We did the
11 commercial dedication in-house. We will get into some more
12 detail as we go through it. At this point I want to
13 introduce Mike Brothers, who is the Systems Engineering
14 Supervisor at Connecticut Yankee.

15 [Slides.]

16 MR. BROTHERS: The reactor protection system, I
17 want to talk about the reactor protection system, aux
18 feedwater and feedwater. I want to do it pretty quickly. I
19 do want to address what you asked before. Really, what I
20 want to talk about in listening this morning, is some of the
21 stuff that you guys were talking about in terms of the
22 complexity of the system, et cetera.

23 The reactor protection system upgrade at CY is
24 really an incremental step. We are half way between the
25 analog systems that were previously existing even in

1 pneumatic systems and the digital protection systems that
2 you guys were talking about this morning.

3 What we used throughout for the reactor protection
4 system phases 1, 2 and 3 were this type of device, okay.
5 This is a Spec 200 micro card manufactured by Foxboro. The
6 closest thing that this is, it's like a very simple hand
7 held calculator. There is no digital communication off this
8 board. Everything that comes off this board is analog.
9 Everything that comes into this board is analog.

10 The problems of communication that you have talked
11 about for the more advanced systems really don't exist.
12 It's a single purpose type of device. It's a configurable
13 system but I said programmable system. What we use, we have
14 currently installed, 103 of these cards. Feedwater control
15 system, we will install about 70 more. So, we will be up to
16 about 173 Spec 200 micro cards. That's the double card.
17 They also come in a single version.

18 MR. LEWIS: Is that robust enough to be passed
19 around the table?

20 MR. BROTHERS: Sure. It's been all around. It's
21 not qualified, so you can do whatever you want with it.
22 Phase I was the front end work. What I mean by front end
23 work was the channels. The majority of the channels up to
24 the existing relay matrix, that was the transmitter, the
25 analog portion of the loop and the portion where the actual

1 trip signals, the input to the existing trip matrix were
2 developed.

3 What we are talking about there is pressurizer
4 level, pressurizer pressure, containment pressure, that type
5 of device. Also, the temperatures, T-AV and Delta T, they
6 are used in the calculation for variable load pressure trip.

7 One of the things you brought up was the
8 complexity of the system. I would love to get the sort of
9 software engineering horsepower that we have -- at least
10 what I am hearing on this side of the table -- up, and see
11 how simple what we are talking about is. I counted them up
12 while we were talking and there's 59 inputs. That's it, 59
13 inputs from that 13 separate trips are developed. We are
14 not talking about something that is very complicated. We
15 are talking about something that is very simple.

16 The front end work, replace that. It interfaced
17 with the existing relay matrix logic which developed the
18 trips, the two out of four trips that you talked about.
19 That was the volting in the system, if you would, just a
20 hard wire type of system. That started in 1987 and 1988.

21 In 1988 and 1989, phase II, replace the relay
22 matrix now with the same type of hardware in which we
23 designed in which we designed in channels. We have two
24 reactor trip breakers. Either one will trip the plant. We
25 have volting within each one of those trains. There is

1 effectively four channels of logic and they are all
2 identical.

3 Phase III is going to be on this coming outage.
4 That is the protection portion of the feedwater control.
5 That is steam generator level coincident with steam flow,
6 feed flow trips. That's the only trip. We have a steam
7 line break, steam line isolation, signal and the control
8 portion of the loop. In doing the controls we also are
9 adding a lot more robustness. It's much easier to trip one
10 of these plants from the control side than from the
11 protection side.

12 MR. CARROLL: These are motor driven pumps, if I
13 remember right?

14 MR. BROTHERS: Yes, they are. They are motor
15 driven pumps. The auxillary feedwater system, of course, is
16 steam driven. Let me go to aux feedwater here. Aux
17 feedwater, we did last cycle. This is a little different
18 than Phase I, II and III of RPS. This utilizes the Woodward
19 governor system out of Fort Collins, Colorado. They are not
20 an Appendix B supplier. We bought their model 501 digital
21 control systems and performed a commercial grade dedication
22 ourselves of their software.

23 They didn't know what ANSI 7-4.3.2 was. They
24 didn't know any of that. What we did is, we received a very
25 good audit from the NRC, a very practical type of audit,

1 looking at the complexity of the system

2 We certainly did our best to prove that would the
3 simplicity of the system that you have here on each Terry
4 Turbine you have speed input from a magnetic sensor -- you
5 have steam inlet pressure and discharge pressure. That's
6 the only parameters they are controlling. That is used to
7 control the speed of the turbine, normally running at a
8 default speed of 4,500 RPM. That's really all there is to
9 it.

10 We took that system -- this is, in fact, a single
11 computer doing all of that. There's a single computer for
12 each of the Terry Turbines. There are two Terry Turbines.
13 That's just a brand name of the steam driven pump that pumps
14 aux feedwater to the system in the event of loss of AC.

15 MR. LEWIS: Just as a minor point of
16 clarification, you said this was analog in, analog out. I
17 couldn't read many of the chip numbers, but I saw some TTL
18 in there.

19 MR. BROTHERS: Inside. You are correct, there is
20 TTL. The TTL is the contact to logic. It's a dry contact,
21 is what it is taking input.

22 MR. LEWIS: If I understand you, you are using the
23 TTL as selector of the analog systems.

24 MR. BROTHERS: Yes. I didn't mean to -- by
25 analog, I meant not microprocessor. I guess I should have

1 said it. There's both dry contact and analog coming in.

2 Aux feedwater is determined to be an unreviewed
3 safety question.

4 MR. LEWIS: I didn't understand that last thin.

5 MR. CATTON: What does dry contact mean?

6 MR. BROTHERS: Dry contact is a mechanical
7 contact. When we say dry contact we mean like a relay,
8 physical break, a contact which opens up and which there's a
9 physical break. A non-dry contact is like a transistor that
10 sinks to ground.

11 MR. LEWIS: But I didn't understand the comment
12 about microprocessor. TTL isn't a microprocessor.

13 MR. BROTHERS: Absolutely not. I agree with that.
14 I am trying to say that the inputs to the card are either
15 one of two types. They are contacts or they are voltage
16 signals. That's the only thing coming into that card.

17 MR. LEWIS: They are analog.

18 MR. BROTHERS: Yes.

19 MR. LEWIS: The TTL is used as a switcher among
20 the analog --

21 MR. BROTHERS: Exactly. The aux feedwater was
22 determined to be an unreviewed safety question. This had
23 nothing to do with the upgrade of the microprocessor,
24 however. It was because of the failure mode of the steam
25 valve. That valve previously failed open. We thought for a

1 long time that was a safe mode that the pump would run, but
2 it was found -- as Jerry talked about -- that it would in
3 fact over speed.

4 The valve was changed to fail closed. That made
5 it an unreviewed safety question. We got prior approval,
6 and prior to start up we had an audit by the NRC which
7 accepted our -- with some provisions that we follow on with
8 our software requirements specification that we developed -
9 -that it was allowed to start up. That's running now. It's
10 been a good upgrade.

11 [Slides.]

12 MR. BROTHERS: Jerry talked about some of the
13 reasons that we did the upgrades, obsolete equipment, that
14 pretty much speaks for itself. The feedwater system that is
15 in there right now is original equipment, 1964 I guess is
16 when it went in. In 1968 the plant started up. It's been
17 there for a long time. That's Hagen equipment.

18 The older equipment that we replaced with RPS was
19 Foxboro H Line. In the case of auxillary feedwater system
20 that was effectively just a control error system. There was
21 only a manual loader on that, no controls of any sort.

22 In addition to the obsolete equipment we had other
23 requirements that were going on, Appendix R. We were forced
24 to comply with that. That required additional channels that
25 dovetailed nicely with the RPS upgrades. Nuclear

1 instrumentation system was replaced. That's not a
2 microprocessor based system. However, with knowing what we
3 are going to do in RPS Phase II, the logic part, it fit
4 nicely to go ahead and replace the NIS and RPS together such
5 that we could duplicate the logic and make them fit
6 correctly.

7 ISAP is our integrated safety assessment program
8 which gives a ranking or a benefit to cost ratio, and the
9 higher the number the better. RPS and NIS came out with
10 fairly high numbers. Once again, they were justified in
11 replacing them.

12 [Slides.]

13 MR. BROTHERS: Let's do feedwater. When I get
14 done with feedwater we will do the last page there.
15 Feedwater upgrade, this continues RPS. Safety related and
16 control system upgrade as was mentioned earlier. We added
17 channels. Presently it is not single failure proof or
18 robust from the standpoint of avoiding an inadvertent
19 reactor trip due to a failure. We have two steam flow, two
20 feed flow, three narrow range levels. We previously had one
21 of each.

22 We also added a primary and a secondary controller
23 on the control side. The control side is much more
24 complicated than the protection side, for feedwater. The
25 protection side carries on the philosophy of RPS Phase II,

1 and has four channels which feed the logic racks that were
2 put in to develop the trips in the steam line isolation.

3 Once again, we will be using the same type of
4 equipment, Spec 200 Micro. We have a couple of other notes
5 here, Rosemount Transmitters were -- you may know that
6 Northeast Utilities has had kind of a special relationship
7 with Rosemount for some years.

8 MR. CARROLL: So have a lot of other people.

9 MR. BROTHERS: We are doing that anyway. We feel
10 it's the right thing to do. We are putting Rosemount
11 transmitters in, a better transmitter. We are taking the
12 opportunity to upgrade to Rosemount.

13 We are asking for prior approval. Even though
14 it's not an unreviewed safety question, we have a submittal
15 into the NRC. That's in keeping with the SER that we got
16 for RPS Phase I and Phase II which I will talk about in a
17 second.

18 [Slides.]

19 MR. BROTHERS: Feedwater control, as I said,
20 primary and secondary system for fault tolerance. We have a
21 median select for level. That solves the IEEE 279 control
22 protection interaction situation for three narrow range
23 levels and deviation alarms for the other channels.

24 MR. CARROLL: Tell me what median select means.

25 MR. BROTHERS: Three narrow range levels, okay.

1 The operator can select any of the three to control from.
2 However, if any of the three deviate by any of the other two
3 by more than five percent it will pick the middle one
4 automatically, is what will happen. I didn't explain that
5 very well and hardly understood it myself when I said that.

6 Did you get that?

7 MR. CARROLL: Yes, I got it. I know what you are
8 talking about.

9 MR. BROTHERS: I am glad, because I kind of
10 confused myself there.

11 [Slides.]

12 MR. BROTHERS: Method of upgrade, that's what I
13 want to talk about. We did this RPS Phase I, II and III
14 under the 50.59 process. We did receive a SER on RPS Phase
15 I and Phase II, and I have in there with caveats. The
16 caveats were that although the NRC found our upgrade to be
17 safe and they agreed with the normal conclusions we should
18 have gotten prior approval anyway. That was really what we
19 got out of it.

20 It was unclear and the reason we are still here is
21 because it's unclear. The draft generic letter is coming
22 out that Jerry is going to be talking about in a little bit.
23 It gives a little more guidance. Effectively what they are
24 saying and what we are saying is, simply because it's an
25 upgrade from analog to digital doesn't make it an unreviewed

1 safety question on its own.

2 I wanted to talk about FMEA which we did. We took
3 advantage of the fact and did address the fact of the
4 channels that we have. Although each of the cards within
5 the four channels typically is the same type of card running
6 the same type of software, the channel separation itself and
7 the design of the plant led us to address every failure mode
8 we felt of the cards; fail low, fail high, fail as is, power
9 interruption, EMI. We addressed each of these situations.

10 We conclude it was not an unreviewed safety
11 question. I think that there was some talk at some time
12 that any digital upgrade is an unreviewed safety question
13 and we don't agree with that, particularly in the
14 incremental approach that we are having right now. I am not
15 being facetious when I say I would like for you to come to
16 the plant.

17 What we have is so simple in comparison to what
18 you guys are talking about, that it really is possible if
19 you were to take a dedicated system and design it from
20 scratch to do something as simple as we are talking about
21 for safety applications, you could in fact do it. The
22 problem is that most of these systems that we are
23 considering and most of the vendors are marketing are
24 systems that were originally developed for the industrial
25 applications.

1 Spec 200 micro was developed in 1981 by Foxboro
2 and spent about three years out there in refineries and that
3 sort of thing going along, doing its thing. They said we
4 can upgrade this and we can qualify it. That software on
5 board that card is a simple rudimentary operating system.
6 That micro the way it works is, that card has effectively 21
7 different blocks that you can access when you configure that
8 card.

9 Each of the blocks is in Rohm, and we access those
10 particular functions, alarm block PNID block, that sort of
11 thing. That software that is held in the Rohm for each of
12 those blocks is what the NRC audited Foxboro on. That's
13 where they did their V&V. They didn't have a V&V program
14 when they built this stuff. They backfitted it like most
15 people did, once they said we want to try to sell it and try
16 to qualify it.

17 They went through some growing pains and the NRC
18 did in auditing them, but the final result was that it's
19 acceptable. It's been out there running for ten years. I
20 think IBM said when they started looking into software
21 reliability, they said that one of the most true methods of
22 working the bugs out of the system is actually have it out
23 there and let it run in applications.

24 The applications we are talking about running on
25 these PC's or mainframes are so much more complicated than

1 what we are talking about there that it's just laughable to
2 sit and worry about can I write code that is going to change
3 stage when pressure in level goes from 82 to 83.5 percent.

4 It's a joke. It's so easy that I don't understand
5 -- what I am trying to get at is that sometimes we make it
6 harder than it has to be, particularly for the stage we are
7 at right now. I agree, that as we go to the integrated
8 protection and control systems that are proposed in the
9 advanced reactor design, that we are going to need to look
10 at those issues particularly when you have one single
11 computer doing multiple functions. We don't have that right
12 now, not at CY. Other people do have those.

13 If we make it as hard as what it's sounding like
14 it is going to be, we are going to be out of business. We
15 won't have a job at CY, the plant won't run, and we won't
16 have anybody to come up and audit. What I am getting at is,
17 don't make it harder than it has to be. There are no analog
18 upgrades out there. We have to replace this equipment. No
19 vendor that I am aware of makes qualified analog
20 replacements anymore so here is where we are at.

21 I am at the plant. I have a plant that needs to
22 run. I have to replace this equipment. I got to replace it
23 with digital equipment and I don't have any guidelines.
24 That's where we are at right now. Everybody is holding on
25 as hard as they can, trying not to get in the situation we

1 got into at CY. We are able to run and we have run very
2 well. The upgrade has been successful. I think we have
3 proven that we can do a good job, and I think most of the
4 other plants that are out there have proven the same thing.

5 The complexity that we are talking about in
6 comparison to even simple computer games is tinker toys in
7 comparison to those games that are out there.

8 MR. CARROLL: Amplify a little bit on the
9 "situation you got into at CY." What does that mean.

10 MR. BROTHERS: In 1988 we had been going via our
11 resident inspector in the region with the NRC on the
12 upgrade, RPS Phase I. As we were getting ready to start up
13 we got a call from the staff at that time here, in Bethesda.
14 They effectively said that they consider it a start up issue
15 and we weren't starting up.

16 We went down in force down here and were able to
17 convince them that it was. We went ahead and started up.
18 We received the SER. We were already in the process of
19 doing RPS Phase II when we got the SER for Phase I. That's
20 the situation we got into. There was a lot of talk amongst
21 other utilities and other vendors as to how did we do what
22 we did. We did it, mainly because I figured it was easier
23 to do it and explain it than to get permission.

24 If we waited to get permission we never would have
25 done it. We did it, and we didn't feel like it was a safety

1 issue. We didn't feel like we violated any of the rules.
2 We addressed all of the questions, the three questions in
3 50.59, and we stand by it. That's what I mean by the
4 situation we got into. I don't think I have anything else
5 to say.

6 I am going to turn it over to Jerry Van Nooredenn,
7 to talk about the comments on the draft generic letter.

8 [Slides.]

9 MR. VAN NOOREDENN: Thank you, Mike. I have a few
10 comments that we have been discussing at Northeast Utilities
11 that I wanted to make on the draft generic letter. First of
12 all, we support NUMARC's efforts as Tony Pietrangelo
13 discussed, the current efforts that are going on with the
14 licensing committee. I am a member of that committee, and
15 we wholeheartedly support all the work that they are doing
16 with the staff.

17 One point that was brought up this morning that I
18 found very interesting, was the emphasis on the design and
19 the specification. We agree, that common mode software
20 failure we feel for these simple systems especially in this
21 application as Mike discussed, it's really a design issue.
22 For a simple situation that we have going on at Connecticut
23 Yankee, if the vendor does an adequate V&V common mode
24 failure should not be a concern.

25 This is a point maybe where utilities are not in

1 the software business. We are in the business of buying
2 these upgrades from the vendors. It will probably be many,
3 many years before a utility attempts to design an install
4 its own system. I doubt if it will ever happen, because
5 it's much easier to buy a product from a vendor and install
6 it that way.

7 The third item is as Mike discussed, what we focus
8 on is the system response in determining the failure modes
9 and effects analysis. In this simple system we don't look
10 at whether or not a resistor or transistor is going to fail
11 in that card. We look at whether or not the card is going
12 to work or not, and is it going to fail high/fail low, what
13 is it going to do when it loses power. We try to keep
14 things on a very simple level.

15 In the types of upgrades that are going on now in
16 the industry if the focus is on that failure mode and
17 effects analysis, I think that will resolve a lot of the
18 concerns that people have. In order for us to do the 50.59
19 we had to come to the conclusion that the software failure -
20 - by that I mean the common mode setup software failure --
21 was not a credible event in a simple system that we have
22 here.

23 We felt pretty good about one, from the many years
24 of industrial operating experience that this operator had
25 not only in the nuclear industry but also in the fossil

1 industry and the chemical industry and the petroleum
2 industry. There are many, many thousands of these cards in
3 operation for going on two decades.

4 The other point is that there is enough on this
5 specific manufacturer of this product -- there is enough
6 experience out there, that you can come up with some
7 reliability numbers and some failure numbers. In fact, we
8 have included those numbers in our submittal that we just
9 made to the staff on our pending upgrade.

10 MR. CARROLL: There have been failures of one sort
11 or another?

12 MR. VAN NOOREDENN: Yes.

13 MR. CARROLL: In this experience base. What is
14 the worst one that happened, or what would its effect had
15 been if it been in a nuclear power plant application?

16 MR. VAN NOOREDENN: We have experienced two cards
17 that have had a failure at Connecticut Yankee since we first
18 installed these, and both times the card flagged itself and
19 set off an alarm. We knew right away that there was a
20 problem. I think it was a hardware. Mike, do you recall?

21 MR. BROTHERS: Yes. In answer to your question
22 the effect it would have on the plant is really what I was
23 trying to say. Jerry is doing a better job of it than me.

24 With one card failure no single card failure can
25 affect or stop a needed safety function. A single card

1 failure can cause an inadvertent actuation of control system
2 which could lead to the requirement for a protection system
3 action. That is addressed and covered under the old IEEE
4 279 and IEEE 603.

5 What I am saying is that the effect -- the two
6 card failures that we had were negligible. They failed as
7 predicted, and we replaced the cards. One was a cold water
8 interlock that would have prevented the opening of a loop
9 stop valve with a Delta T too high. The other one was
10 associated with the logic portion of the reactor coolant
11 system low flow.

12 Once again, individually in those cards there's
13 three other channels doing that logic. The answer is that
14 it was negligible, in our experience.

15 MR. CARROLL: The system design is such that you
16 always know that a failure of that nature has occurred.

17 MR. BROTHERS: We rely upon the surveillance
18 interval to pick up any undetected failures. That's what we
19 based our surveillance interval on. We do six week
20 surveillance of all the protective functions.

21 I cannot say without question that every single
22 failure will be picked up, but that's the basis for the
23 surveillance interval that we have.

24 MR. CARROLL: In principle at least, you could
25 have one card failed and you don't know about it and then a

1 second one fails.

2 MR. BROTHERS: Right, and then you still can make
3 two out of four out of the other four.

4 MR. CARROLL: Then, a third one fails.

5 MR. BROTHERS: That's the interval that we have,
6 is to preclude that from happening. It is determined that
7 it is based upon the information that we have from a vendor,
8 the justification for the surveillance interval that that,
9 in fact, will not happen. Only a maximum of one single
10 undetected failure will miss a surveillance, if you know
11 what I'm saying. That's the basis for our interval.

12 MR. CARROLL: That's based on the manufacturer's
13 database of experience not only the small experience you
14 have but the broader experience that --

15 MR. BROTHERS: Right, that's correct. In fact, we
16 surveil it at a much higher frequency than what the
17 manufacturer recommends. It was easier for us to do that
18 and keep it with the old surveillance interval rather than
19 try to justify a newer one.

20 We are involved with some owners group activities
21 which may reduce that.

22 MR. CARROLL: Thank you.

23 MR. WYLIE: Let me ask a question. When you
24 decided to go to this change did you have to do anything to
25 your grounding and shielding system for the inputs and

1 outputs?

2 MR. VAN NOOREDENN: Connecticut Yankee has an
3 older grounding system, unlike the newer plants out there.

4 MR. BROTHERS: We don't have separate instrument
5 or power supply grounds at Connecticut Yankee. We looked at
6 it, checked with the vendor once again for the requirements,
7 the stability of the power supply. Remember, each of the
8 cards has its own on board five volt power supply. That
9 five volts is derived by a 15 volt rack power supply which
10 is coming from our station, 120 volt supply which is
11 inverter driven.

12 That's the stability of the grid, 120 volt supply
13 to that. The grounding has never been an issue for us. We
14 have not looked at it. We addressed it, via our response to
15 the EMI initiatives.

16 MR. VAN NOOREDENN: The last point I wanted to
17 make was something we feel that is also necessary to --
18 somewhat to address your point of undetectable failures.
19 That is for the reactor protection system and engineered
20 safety features, actuation systems that a defense in depth
21 analysis recommended. That is something that we did, what
22 if this thing failed when you had a LOCA and loss of offsite
23 power, what would happen.

24 We tried to look at what other systems, what other
25 trips would come in and still bring the reactor down and

1 bring it to safe shutdown. It was somewhat qualitative but
2 it's an exercise that we went through, and we thought it was
3 very useful to do something like that. If you are talking
4 about replacing a rad monitor or something, I don't think
5 you have to do this.

6 [Slides.]

7 MR. VAN NOOREDENN: Some other comments on the
8 draft generic letter. One of the surprises that we
9 encountered back in 1987 when we did the first phase of the
10 reactor protection system upgrade, the front end work was on
11 EMI and RFI levels. We really didn't factor that into our
12 project description. When those questions were raised we
13 went back and looked at the installed locations.

14 These cabinets and these cards are all in the
15 control rooms, so they are not out in the plant. The one
16 obvious thing that could give you a problem of the walkie
17 talkies, the portable hand held radios, the way we have
18 taken care of that -- just like you would for the normal
19 digital system is -- you don't allow those in the control
20 room. By procedural controls you eliminate that source of
21 EMI/RFI.

22 Our experience has found that the field that we
23 have measured were within the design parameters that the
24 manufacture specified for his equipment. We haven't found
25 any surprises out there. We do feel that reasonable

1 standards are needed. There are some standards out there.

2 The administrative controls like for the walkie
3 talkies, hand held radios, we think those are acceptable and
4 especially in a place like the control room. We have
5 recently encountered an upgrade, done something on a much
6 smaller scale, at Millstone III. We don't feel a site
7 specific survey should be required for all applications.

8 If you have some baseline data, every time you do
9 an upgrade you shouldn't be required to automatically go
10 back and re-verify everything from a few years earlier if
11 nothing has really changed. If you are going to stick a
12 system next to a rotating piece of machinery or something
13 like that, you better survey it and find out if you have a
14 problem.

15 If you have some baseline data, hopefully, that
16 would be good enough. I know there's lots of discussion
17 going on in the industry in that effort, and EPRI has a
18 large effort going on in that area.

19 In summary, I wanted to say our experience with
20 the A to D conversions has been excellent. As we said, we
21 have only had two failures on the cards which were flagged
22 right away. So, the operators knew right away that there
23 was a problem. We encourage the staff to continue the
24 dialogue with NUMARC and to reach a position where the
25 licensing process is stabilized. The process at the present

1 time of looking at what the last SER is that has been issued
2 by the staff is a volatile process, it's a moving target.

3 If there is some published criteria, defined
4 criteria that everybody can follow, that would be very
5 helpful.

6 Again, reasonable standards and guidance are
7 needed, and we definitely feel that is the case. We want
8 the staff and the ACRS to continue on the efforts.

9 The last point is really to let the licensee
10 decide when the conversion is a USQ. It's basically
11 business as usual. We do that for every other change that
12 we make to the plant, and we don't feel that the upgrades
13 using digital upgrades and using software should be any
14 different.

15 MR. WYLIE: Have you experienced any spurious
16 operation of these things?

17 MR. VAN NOOREDENN: Any spurious operation, no.

18 MR. WYLIE: Tripping of the unit?

19 MR. VAN NOOREDENN: No, we haven't.

20 MR. KEMMERER: Jerry, you said you don't write
21 software but you buy it from the vendors.

22 MR. VAN NOOREDENN: That's right.

23 MR. KEMMERER: How does the level of the software
24 you buy compare to what Mr. Cobb was calling components.
25 It's at the component level?

1 MR. VAN NOOREDENN: I think Mike is able to handle
2 that.

3 MR. BROTHERS: To be honest with you, in the case
4 of Woodward governor we had direct involvement with the
5 configuration of that control. That's the auxillary
6 feedwater system. We did have some involvement in the final
7 design of that software. We didn't write the code.
8 Everything once again there is in a E-prong type. We simply
9 downloaded the configuration. We had nothing to do with the
10 operating system for their computer.

11 Foxboro, there, we were required to participate in
12 an audit in 1987 and 1988 for their source code that I
13 talked about on the Rohm devices on that card. We have had
14 no input. They have done some upgrades based upon some
15 findings that they have had and some improvements they have
16 made, but we don't have access to source code nor do we want
17 access to source code.

18 It probably is being treated by us as a component,
19 yes.

20 MR. LEWIS: Are there further questions?

21 [No response.]

22 MR. LEWIS: Thank you, very much. I guess the
23 next item on our agenda is to hear from Southern Cal Edison
24 on the same subject. I have to plead a conflict of
25 interest, in the sense that they supply the electricity to

1 my house, and don't do it very well. In fact, I have to say
2 further, that the failure of their network caused my
3 computer to fail and not the other way around.

4 [Laughter.]

5 [Slides.]

6 MR. PLAPPERT: Good afternoon. My name is Bob
7 Plappert, representing the Nuclear Organization of the
8 Southern California Edison Company today. Most of you know,
9 but I will mention it anyway, SCEC operates slightly greater
10 than two nuclear power plants presently, at the San Onofre
11 nuclear generating station on the Southern Coast of
12 California.

13 My job presently is an engineering group
14 supervisor in the controls discipline, my primary
15 responsibilities being in the area of radiation monitors.
16 One of the issues that myself and the group are responsible
17 for is responding to day to day concerns with radiation
18 monitors at the plant that may have some effect on continued
19 operations, to longer term issues having to do with design
20 changes, regulatory requirements, both present and emerging
21 requirements, and maintaining the consistency of the RMS,
22 radiation monitoring systems with the plant design basis.

23 It's interesting for me to follow the presentation
24 given by Jerry. They went through some length to convince
25 the Committee that the upgrade that they described was very

1 simple and straightforward. If they were successful in
2 convincing you that that was simple and straightforward, I
3 will have no problem convincing you that this upgrade that I
4 am going to describe is two or three orders of magnitude
5 more simple.

6 MR. LEWIS: Are you going to quantify that as we
7 go along?

8 MR. PLAPPERT: I think I just have. Two to three
9 orders of magnitude, how is that.

10 MR. LEWIS: What you mean is that what you show
11 us, I have to assume is two to three orders of magnitude
12 more complicated.

13 MR. PLAPPERT: I think you will agree with that.
14 The purpose of my presentation today is to provide an
15 example to the Committee of, as I said, a very simple and
16 minor upgrade of control system, from an analog to digital
17 system, how it was brought about by SCEC, engineered and
18 evaluated for safety, and our plan for implementation of the
19 upgrade into the plant. We haven't gotten to that point
20 yet.

21 My intent is for the Committee to derive an
22 appreciation of the rigor that we apply to this process,
23 again, a very minor upgrade, and the experience be
24 considered in the development of the ongoing discussions
25 related to NRC staff involvement in the issue of unreviewed

1 safety questions on minor digital upgrades.

2 MR. CARROLL: How would you characterize the
3 staff's present position on that subject?

4 MR. PLAPPERT: Uncertain, which I believe is a
5 good thing to have right now.

6 MR. CARROLL: It's uncertain as to --

7 MR. PLAPPERT: I don't think the staff has made a
8 decision on whether or not they are going to require any
9 digital upgrade of a safety related system to considered an
10 unreviewed safety question.

11 MR. CARROLL: That's what I think, too.

12 [Slides.]

13 MR. PLAPPERT: My presentation will consist of
14 those sections as shown on this transparency. I will first
15 provide a brief background of our radiation monitoring
16 system, complete with simple block diagram. I will focus on
17 the analog to digital upgrade.

18 The reason for the upgrade will be provided,
19 followed by a somewhat detailed presentation of the
20 engineering that was applied, qualification program that was
21 developed, and the testing that the modified components were
22 subjected to. Our plans for implementation will then be
23 discussed. The impact of the change for some of our
24 licensing documents, and there's a couple of other upgrades
25 that we are contemplating at San Onofre, again, simple in

1 nature, that I will briefly mention.

2 Background, there are about 45 channels of process
3 and effluent monitoring systems that are going to be
4 affected by this upgrade. The process systems are typically
5 safety related systems, having to do with isolation of a
6 couple of buildings in the event that we achieve a high
7 radiation level. Containment, purge isolation system is
8 affected, the fuel handling building isolation system is
9 affected, and the control room building isolation system is
10 affected.

11 The majority of the monitors, however, are in the
12 category of non-safety related but important to safety.
13 Those are the radioactive effluent monitoring systems, both
14 liquid and gaseous systems.

15 The engineered safety features or the safety
16 related systems are governed by our technical
17 specifications. They have limiting conditions for operation
18 and accident requirements. Their functionality and
19 reliability can have an effect on our ability to continue
20 plant operations.

21 The effluent monitoring systems, although not
22 governed by the tech specs, they are governed by what we
23 call a licensee control specifications or our ODCM, offsite
24 dose calculation manual. The importance that they play is
25 their one line of defense to ensure that the Federal

1 regulations having to do with the concentrations of
2 effluents in normal discharges from the plant are not
3 exceeded.

4 We have both safety related and non-safety related
5 systems being affected by this upgrade.

6 [Slides.]

7 MR. PLAPPERT: Here, we have a typical block
8 diagram of our MS system. I would like to briefly describe
9 its function and in so doing, point out the changes
10 resulting from the upgrade. There are two basic components,
11 the detector which is located in the field which senses the
12 parameter to be measured, and the control room module which
13 is located in the control room that houses all the
14 electronics which process the incoming frequency from the
15 detector, converts it to a voltage in the current driving a
16 meter recorder for historical and trending purposes, and an
17 alarm card which provides annunciation in the control room
18 actuation for those systems that satisfy a safety function
19 and isolation of the effluents if it happens to be an
20 effluent monitoring system.

21 The component of interest in this discussion is
22 the CRA card shown surrounded by the bubble in the middle.
23 The output of the signal conditioning board which is also
24 shown, is fed into the CRA card which consists basically of
25 an integrated capacitor which produces an apparent direct

1 current proportional to the incoming frequency, a
2 logarithmic element which produces an output voltage which
3 is proportional to the logarithm of the input frequency.
4 It's as simple as that, the function that is performed.

5 The output is the amplified, passed through a
6 precision voltage to current converter and finally driving
7 the meter chart recorder and the alarm card.

8 MR. LEWIS: How does it calculate logarithm?

9 MR. PLAPPERT: Zenor diode which is shown by that
10 symbol, which is operated in the logarithmic portion of its
11 voltage and current characteristic curve.

12 MR. LEWIS: It doesn't have to be a Zenor. Any
13 diode --

14 MR. PLAPPERT: This happens to be a zenor diode
15 which as I will mention in a few minutes, is one of the
16 reasons for this upgrade.

17 MR. LEWIS: In any case it's an analog way to
18 calculate.

19 MR. PLAPPERT: Yes, sir.

20 [Slides.]

21 MR. PLAPPERT: The upgraded configuration looks
22 very similar to what I just showed you, the singular change
23 being in the CRA card which is now referred to as the CRA
24 micro card, which employs microprocessor technology. All of
25 the inputs to and outputs from the CRA micro card are

1 unchanged in this modification.

2 The upgraded CRA micro card utilizes a
3 microprocessor and software to perform the following simple
4 functions. It determines the frequency of the incoming
5 signal, it selects one of two accounting modes to determine
6 what -- depending on the magnitude of the frequency,
7 computes the logarithm of that incoming frequency and
8 converts it to a voltage and a current which is proportional
9 to the logarithm of the incoming frequency.

10 This output drives the meter, the recorder and
11 alarm cards. It will give you actuation and annunciation in
12 the same way as was performed by the analog version.
13 Therefore, the method of measurement of the incoming signal
14 has been transferred from a analog process to a digital
15 process with this simple upgrade.

16 MR. KEMMERER: There's one per detector?

17 MR. PLAPPERT: That's right. Each of the 45
18 channels contains an upgrade, as I have described.

19 MR. LEWIS: Forgive me for being stupid. You lost
20 me somewhere in the circuitry. Before we had some kind of
21 counter which was producing pulses whose frequency was a
22 function of the irradiation. That was going through a
23 zenor, an average and then a zenor diode which converted it
24 to logarithm fully analog, and that produced the signal that
25 drove presumably the meter and the alarms and that sort of

1 thing.

2 MR. PLAPPERT: Yes, sir.

3 MR. LEWIS: At this point, are we actually
4 measuring the frequency or doing the same thing; that is, do
5 we still have the same signal coming in, and what happens to
6 it first.

7 MR. PLAPPERT: Yes. The same signal comes into
8 the CRA micro card as did to the CRA card with the analog
9 version.

10 MR. LEWIS: What happens to it first?

11 MR. PLAPPERT: It enters a counter, a programmable
12 counter which determines two things.

13 MR. LEWIS: It actually counts.

14 MR. PLAPPERT: It counts incoming pulses.

15 MR. LEWIS: Okay, fine. The output of the counter
16 is converted to a voltage, and that voltage, the logarithm
17 is taken how?

18 MR. PLAPPERT: The software computes the logarithm
19 of the counted frequency.

20 MR. LEWIS: The counter frequency is actually
21 produced digitally and then there's a logarithmic converter
22 that converts it to a --

23 MR. PLAPPERT: There is a software algorithm that
24 computes the logarithm of those counts.

25 MR. LEWIS: That sounds an awful lot harder than

1 putting it through a diode.

2 MR. PLAPPERT: I will talk a little bit about the
3 problems with our --

4 MR. LEWIS: I guess I am saying that the signal is
5 going through several forms that it doesn't have to go
6 through. It's your plant, not mine.

7 MR. PLAPPERT: Thank you.

8 MR. LEWIS: Maybe that's why my power failed.

9 MR. PLAPPERT: Don't tell anybody that.

10 MR. LEWIS: Forgive me, I really didn't understand
11 what you were doing.

12 [Slides.]

13 MR. PLAPPERT: Mike had mentioned the reasons for
14 the upgrade, and I doubt if they will surprise any of you.
15 The radiation monitoring system that we have at San Onofre
16 was designed in the 1970's and it's completely analog. It's
17 quite obsolete at this time. As I said, it's more than 20
18 years old.

19 As in many older nuclear power plant systems the
20 availability of qualified spare parts is becoming
21 increasingly difficult, forcing in some cases commercial
22 dedication and expenditure of substantial resources in
23 locating qualified spare parts.

24 One example is the zenor diode, in this case. We
25 found that the zenor diodes that are available now don't

1 satisfy the very strict voltage current characteristics that
2 the original did. This requires us -- they work and they
3 function okay, but it requires us to test a great many
4 number of them in order to find one that is good enough or
5 that satisfies our particular specifications in order to
6 replace one that has been failing.

7 Another example of the obsolete equipment is --
8 not the obsolete equipment but -- we have a great many
9 calibration difficulties. Primarily the reason for our
10 calibration difficulties are the fact that the module is
11 located in a cabinet in the control room environment which
12 is about 30 degrees warmer inside the cabinet than it is
13 outside the cabinet. Calibration process takes place
14 outside the cabinet in about a 30 degree cooler environment.

15 This has required us to build an enclosure,
16 surround the module, apply a heat source, allow it to
17 stabilize in order to do the calibration. This process that
18 I have described takes about 24 hours to perform, causing us
19 to have technicians in our environment in a hallway which is
20 a nuisance. It interferes with operator activities, ongoing
21 normally back and forth. There are so many channels, that
22 there is somebody there constantly. That has been a big
23 problem for us.

24 The calibration should normally take just a few
25 hours if it weren't for this temperature problem. As a

1 result of those two examples and some others, you can see
2 that the maintenance costs in terms of both obtaining
3 qualified spare parts, dedicating spare parts and technician
4 resources, has become unmanageable for us.

5 Emerging from these types of problems are low
6 system availability that we have. It's normally about in
7 the 70 percent range for the total system, both the safety
8 and non-safety monitors. Although we are able to satisfy
9 all the regulatory requirements including our tech specs and
10 OCM requirements, it's difficult to do that sometimes, and
11 it provides unnecessary challenges to the plant operations,
12 especially those that are governed by the tech specs.

13 We have had in the past a great many spurious
14 safety system actuations of the containment purge, fuel
15 handling building and control room isolation as a result of
16 these radiation monitoring system deficiencies. There
17 aren't quite as many now, because we have ironed out some of
18 the noise sources that have been causing them. They still
19 do happen periodically.

20 They are a nuisance to the plant staff and a
21 distraction to the plant staff. Up until recently they
22 prompted reports to the NRC operations center, which we
23 prefer not to have. Finally, the result in unwanted and
24 undesirable challenges to the safety systems when they
25 become actuated.

1 We expect that the upgrade that we hope to put in
2 will provide a substantial improvement in all of these
3 areas.

4 MR. KEMMERER: Is the noise coming from within the
5 bubble?

6 MR. PLAPPERT: No, it's surrounding plant
7 equipment.

8 MR. KEMMERER: Is the new equipment supposed to
9 deal with that? I was trying to figure out how that bullet
10 was handled by putting the microprocessor in there.

11 MR. PLAPPERT: Now that I think of it, I am not
12 sure it will. As I said before, that doesn't seem to be a
13 big a problem as before.

14 MR. WYLIE: In addition to putting your
15 microprocessor in there, are you doing anything to your
16 grounding system, or are you using the same one you had?

17 MR. PLAPPERT: We have made a lot of improvements
18 on our grounding system to solve the problem of the spurious
19 actuation problems with the analog system. There aren't any
20 plans to pursue additional improvements. If we get
21 experience with the new card which results in continued
22 spurious actuations, we will have to look at that as well.

23 MR. CARROLL: Just out of curiosity, do you know
24 why the characteristics of the zenor diodes changed? What
25 does the vendor say?

1 MR. PLAPPERT: I don't know.

2 MR. CARROLL: They have just changed.

3 MR. PLAPPERT: It must operate in a very narrow
4 section of the logarithmic curve on the characteristic. If
5 we can't get one to operate properly with that curve, then
6 we found that we don't get the accuracy requirements that
7 the system must satisfy.

8 MR. LEWIS: I was a little surprised that you are
9 using a zenor. To get a logarithmic response you should
10 operate in the forward direction, and zenor is always
11 defined for reverse applications. The temperature
12 sensitivity is certainly relevant. The logarithmic part of
13 a diode characteristic in the forward direction is a
14 consequence of the laws of nature. That simply cannot
15 change.

16 It's a matter of what the reverse current is or
17 something like that, something peculiar. The forward
18 direction in the current density and the temperature are the
19 only parameters that determine the shape of a logarithmic
20 curve. There is still something missing. I share a little
21 bit of Jay's nervousness about this.

22 I can buy zenors and I can buy diodes, and they
23 are the same as they ever were.

24 MR. PLAPPERT: Our experience is that we have to
25 test a lot of them before we can get one that will function

1 properly.

2 MR. LEWIS: It may have been an original mistake
3 using a zenor.

4 MR. PLAPPERT: Perhaps it was. I am going to go
5 just briefly through what we did in the way of engineering
6 and equipment qualification to satisfy ourselves that the
7 newly designed system will satisfy all of the original
8 requirements.

9 [Slides.]

10 MR. PLAPPERT: It was intended that this upgrade
11 be installed in safety as well as non-safety systems, as I
12 mentioned before. So, we had to qualify the entire systems
13 with the original requirements. As far as hardware is
14 concerned I put some of the major ones up here, the IEEE
15 standards having to do with Class 1E electrical equipment.
16 Our qualification plan was based on satisfying these
17 requirements.

18 It was expressed earlier today a couple of times
19 that there is no real strict guidance that we are required
20 to apply to software development for these upgrades. Given
21 that, we chose a set of standards that are currently
22 available right now in order to establish a documentation
23 package which we believe would stand up well under our
24 quality assurance scrutiny as well as nuclear regulatory
25 scrutiny as a documentation package for suitability of the

1 software.

2 We developed a software requirements specification
3 in accordance with IEEE 830, which is a guide for software
4 requirement specifications. In the SRS, the main
5 specifications were set forth as is required. Emerging from
6 the SRS was a software design description, and we developed
7 that based on the guidance provided in IEEE 1016 which is
8 the recommended practice for software design descriptions.

9 In the software design description, our STD, we
10 provide the detailed precise information needed for
11 planning, analysis and implementation of the software design
12 such as it contains flow charts, codes, variable
13 description, et cetera, and the detailed description of the
14 various functions of the software.

15 We developed a software verification and
16 validation program based on IEEE 1012. In it is defined a
17 series of tasks including appropriate inputs and outputs
18 which will ensure that the software complies with the
19 existing SRS, STD, all of the functional requirements and
20 exhibits a high level of reliability.

21 Test plan, we developed a rigorous test plan and a
22 procedure with SCE engineers with the assistance from
23 outside consultants, and performed a testing in a qualified
24 laboratory. For a summary of the testing that we did, the
25 test specimens -- and there were three in this case --

1 underwent a burn-in period for about 100 hours at maximum
2 environmental temperature that it would be exposed to and a
3 maximum load of current and voltage load.

4 We performed functional testing following that, to
5 ensure the card continued to satisfy its accuracy and
6 response time requirements. We aged it to its end of life,
7 both mechanically and in terms of radiation and continued
8 functional testing throughout of all of these processes. We
9 did additional environmental testing at --

10 MR. CARROLL: What is aging to end of mechanical
11 life mean.

12 MR. PLAPPERT: There are some pots on the card
13 that can be adjusted mechanically. We cycle them.

14 MR. CARROLL: That was it.

15 MR. PLAPPERT: Yes. It wasn't very much in terms
16 of mechanical aging, but there was some. The environmental
17 testing, we ran the card through its entire range, both
18 temperature and humidity, and range of the monitor which is
19 from ten to ten to the seventh counts per minute.

20 Throughout, we ensured ourselves that the
21 acceptance criteria in terms of accuracy monitor response
22 time and alarm card response time continued to satisfy the
23 acceptance criteria. Finally, we chose one of those cards
24 and performed a seismic test. We installed it into its
25 module, shook it with the San Onofre earthquake spectrum and

1 did functional testing as well.

2 All of the testing is now complete. We are
3 encouraged by the test results. It appears that all of the
4 acceptance criteria will be sufficiently satisfied.

5 [Slides.]

6 MR. PLAPPERT: We also conducted an FMEA, failure
7 modes and effects analysis which addressed each component in
8 the upgrade. We actually included each piece part that we
9 added to the new CRA micro board in the FMEA. We used IEEE
10 standard 352 for guidance, which is the guide for general
11 principles of reliability analysis of nuclear generating
12 station safety programs.

13 The FMEA was done manually, using appropriate
14 electrical schematics. We analyzed the effect of each
15 postulated failure on the subsequent component in the signal
16 path until the consequence of the failure was determined.

17 For software related failures, the card employs a
18 watch dog timer circuit to detect software failures. The
19 way it works is, it monitors the input from the CRA micro
20 card to a digital to analog converter. It causes an alarm
21 condition to occur, an actuation to take place. If the
22 interval between update from the CRA micro to the DAC exceed
23 a preset time limit -- which happens to be about two seconds
24 in this case -- this feature reduces the likelihood of a
25 software failure, for example, a software lock up from going

1 undetected.

2 All of the components in the upgrade were shown in
3 the FMEA to have acceptable failure consequences.

4 MR. WYLIE: Is that in a fail safe mode, is what
5 you are saying?

6 MR. PLAPPERT: Yes. The system will actuate,
7 alarms will be generated in the case of a software failure.

8 MR. CARROLL: In describing your FMEA you said it
9 was done manually. How do you do it other than manually, or
10 how would one --

11 MR. PLAPPERT: I am not sure I understand the
12 question.

13 MR. CARROLL: It implies that there's another way
14 to do it. You said you did it manually.

15 MR. PLAPPERT: We didn't take advantage of any
16 software programs or anything other than an engineer sitting
17 down and going through the schematics, to determine the
18 failure modes.

19 MR. CARROLL: Okay.

20 MR. LEWIS: I am still trying to visualize the
21 signal flow. You mentioned DA converter. Does that mean
22 the signal flow is from the incoming pulses through the
23 counter, then a digital logarithm followed by a DA converter
24 through the output; is that the signal flow?

25 MR. PLAPPERT: Yes.

1 MR. CARROLL: Who is the vendor of this equipment?

2 MR. PLAPPERT: NMC, Nuclear Measurements
3 Corporation -- of the original radiation monitoring
4 equipment, it's NMC.

5 MR. CARROLL: Who is providing --

6 MR. PLAPPERT: SCE engineered the card, and we
7 have employees and consultants to prepare the software,
8 documentation, SRS, V&V, perform the V&V, perform the
9 qualification testing under their quality assurance program.

10 MR. CARROLL: Somebody is going to manufacture it
11 for you?

12 MR. PLAPPERT: Yes. We have a manufacturing
13 facility within SCE that will manufacture it.

14 [Slides.]

15 MR. PLAPPERT: Briefly, our implementation plans.
16 As I mentioned before, all of the testing is complete. Our
17 consultants are in the process of completing the
18 qualification test report, and the software V&V is being
19 completed. The report is expected in just a couple of
20 weeks.

21 We have also obtained one of the boards -- one of
22 the specimens that was tested to end of life. We have done
23 some isotopic testing of our own in a plant shop and are
24 satisfied with the results. Though our intended way to
25 implement it into the plant is during routine surveillance

1 of these monitors, there is no urgency right now because the
2 systems are hobbling along. They are functional. They do
3 need this upgrade, but they are functional.

4 As we have a need for a routine surveillance we
5 will take out the old analog card and put in the new analog
6 card. There is one or two minor seismic supports for the
7 different card that will also be put in place. We expect
8 that would take about six months, from the time we begin.
9 We expect to finish the design work in the qualification
10 program by April or May, of this year.

11 As far as licensing document changes we do have to
12 change the updated final safety analysis report, in that it
13 contains sufficient detail to describe the present radiation
14 monitoring systems as being an analog process. We will make
15 mention of the fact that it's now a digital process.

16 We don't believe that there are any changes to our
17 technical specifications because the function of the monitor
18 is unchanged. The output characteristics of the system is
19 unchanged, and it shouldn't have any affect on the
20 capability. It will not have any affect on the capability
21 of the system to perform its safety function.

22 MR. CARROLL: You have pretty much made up your
23 mind this is not an unreviewed safety question?

24 MR. FLAPPERT: We performed a 50.59 in this case,
25 and believe it not to be an unreviewed safety question. We

1 believe we were able to answer the questions sufficiently,
2 to conclude that it's not.

3 MR. CARROLL: Has the region or the staff here in
4 Washington agreed with that?

5 MR. PLAPPERT: They haven't seen our 50.59.

6 MR. CARROLL: I see.

7 MR. PLAPPERT: It's an open book. That's the
8 purpose of 50.59, you don't have to show it to them.

9 MR. CARROLL: I understand.

10 MR. LEWIS: Does that decision rest in the region
11 or at headquarters?

12 MR. PLAPPERT: I believe it would be subject to
13 the inspection in enforcement effort. It would review our
14 engineering processes, and this might be one that they would
15 review and challenge us on it.

16 [Slides.]

17 MR. PLAPPERT: The problem that I just finished
18 describing is intended to be a temporary solution to our RMS
19 problems. It will provide substantial improvement but we
20 expect that they will be continued problems in getting spare
21 parts for not only the module itself but the detectors out
22 in the plant, the skids, sample pumps and things like that.

23 We are evaluating a complete RMS system upgrade,
24 all of the RMS systems that are presently available on the
25 market to utilize digital technology.

1 We also have a toxic gas isolation system which
2 functions similar to the control room isolation, in that it
3 isolates control room -- not on high radiation signals but
4 on an accident involving a toxic gas such as chlorine or
5 hydrocarbons. It's also obsolete. We have a difficult time
6 getting spare parts, and we are considering a similar
7 digital card upgrade that I have described here for that
8 one.

9 This essentially completes my presentation today.
10 My intent in this discussion was to present to the Committee
11 an example of the upgrade of an obsolete and difficult to
12 maintain control system, taking advantage of modern digital
13 technology. I hope that I have been able to demonstrate the
14 rigor that we applied to this process for a very simple
15 system.

16 The detail that we provided in the engineering and
17 the qualification program to ensure that the newly designed
18 system satisfies the same criteria as the original system,
19 and will continue to satisfy its safety function and
20 regulatory requirements. I encourage that all of those here
21 consider this minor type of experience in any ongoing
22 discussions related to additional or regulatory requirements
23 or guidance in conversion of analog systems to digital
24 upgrades.

25 I appreciate the opportunity to make this

1 presentation to the Committee, and express my thanks to you
2 for allowing me to do so.

3 MR. LEWIS: As I understand it, this is a system
4 which is really an analog system, input and output but there
5 is some digital intermediate steps. You could have plotted
6 them in a box and never told anybody, and it would have then
7 been an analog to analog conversion which would have met
8 specs; is that right?

9 MR. PLAPPERT: Yes, I guess that's a way to put
10 it.

11 MR. LEWIS: You could have lied.

12 MR. PLAPPERT: Yes. We actually developed the
13 card before we did all the engineering that I have described
14 here. It was a problem, that the spurious actuations, the
15 difficulty in calibration, the problem with the diodes, have
16 been with us for about four years now. We have some pretty
17 bright engineers at the plant who are good at this kind of
18 thing.

19 We said can fix this. It's just this card for
20 this set of problems. The technology is available. It's
21 not very complicated programming. It's not very complicated
22 components. The testing to accomplish this is very simple.
23 Why don't we just go and build this, and we did. We had it
24 built three years ago by what we call shop services lab,
25 another part of the Edison Company.

1 We got it back and tested it at that time and said
2 this is great, let's put it in. Then we scratched our head
3 and said wait a minute, we want to use this in safety
4 systems. There is a whole gamut of things that we really
5 didn't apply to this process when we built it. What we are
6 doing now and what I have described to you is backfitting
7 those processes, the equipment qualification plan, the
8 seismic testing, the development of a software documentation
9 package, so that when we decide to install it into the plant
10 we have what we believe a strong case, a strong
11 documentation trail to satisfy the most critical of auditors
12 or investigators.

13 It's a reverse engineering process that I have
14 described.

15 MR. LEWIS: The only reason you have to take the
16 logarithm is to display on a wide range meter. You could
17 have used auto scaling and just avoided the logarithm
18 completely, but that would have involved changing things
19 that interface with this system. Is that the issue?

20 MR. PLAPPERT: Yes.

21 MR. LEWIS: When you go back to doing the whole
22 radiation monitoring system, you might well just get rid of
23 the logarithm. You don't need it, and it doesn't serve any
24 useful purpose.

25 MR. PLAPPERT: Yes, that's right.

1 MR. CARROLL: It does though, Hal, in the sense
2 that an awful lot of these radiation monitors, you really do
3 want a logarithm scale to see trends and stuff. They can
4 change pretty radically.

5 MR. LEWIS: Yes.

6 MR. CARROLL: Things like auto scaling don't show
7 you the trends that well, do they?

8 MR. LEWIS: You do auto scaling, you can have a
9 little light that tells you which scale you are on.

10 MR. CARROLL: I know. I guess I like to see a
11 strip chart that shows me changes.

12 MR. LEWIS: But you are very old.

13 [Laughter.]

14 MR. KEMMERER: You are using the same card, then?
15 You are using the card that you developed prior to doing the
16 work?

17 MR. PLAPPERT: Yes. We built the card first, and
18 then a term that we are using -- I don't know if it's an
19 industry-wide term -- we are reverse engineering the -- we
20 are doing the engineering after the development of the card
21 to support its use in the plant.

22 I actually brought the old card and the new card
23 with me, if anyone would like to see it. I can pass it
24 around.

25 MR. COBB: Would your costs have been less and

1 your process faster, if you had done it -- not doing the
2 reverse engineering, if you had followed the right process.

3 MR. PLAPPERT: Yes. We have expended far more in
4 terms of dollar and engineering resources to use this card
5 than we expected to do.

6 MR. KEMMERER: Do you have a worse product?

7 MR. PLAPPERT: No. Do we have a worse product?

8 MR. KEMMERER: A worse product.

9 MR. PLAPPERT: With the upgrade?

10 MR. KEMMERER: Following the other half of his
11 question -- what we heard this morning was due to planning
12 up front get a better product.

13 MR. PLAPPERT: No.

14 MR. KEMMERER: The reason you took longer, I want
15 to know if you got a worse product.

16 MR. PLAPPERT: This is inconsistent with that. We
17 built it first and are coming behind it with documentation.

18 MR. WYLIE: What are your future plans for
19 extending the upgrade?

20 MR. PLAPPERT: Extending the upgrade, do you mean
21 upgrading more components?

22 MR. WYLIE: Right.

23 MR. PLAPPERT: There is an alarm card which is an
24 analog card that uses the bistables, that we have also
25 developed a digital alternative to. We want to finish this

1 one first before we go onto that, because we would have to
2 apply the lessons learned from this experience will
3 hopefully cause that one to be far less painful.

4 MR. CARROLL: You keep doing this sort of thing
5 and you will ultimately end up with your brand new --

6 MR. PLAPPERT: A brand new system, yes. When we
7 sit down and brainstorm on the new system we considered
8 whether, are we half way to a new system with this upgrade.
9 We don't think so, yet.

10 MR. LEWIS: Thank you very much.

11 MR. CARROLL: I guess I have one other question.
12 Do you know if there are other utilities that would
13 undertake a project of this sort, to the extent of actually
14 designing and manufacturing their own equipment?

15 MR. PLAPPERT: I don't know of any. That doesn't
16 mean that there aren't. I just don't have a lot of
17 experience with what other people are doing.

18 MR. CARROLL: I don't think most utilities would
19 do that. They would go and get a vendor to do it for them.

20 MR. PLAPPERT: We have gotten a vendor to help us
21 with the documentation, the STD and SRS and so on. If we
22 had approached it from engineering first and manufacturing
23 second, we may have had it all done by a vendor.

24 MR. LEWIS: Thank you. We have heard from the
25 Northeastern part of the country and the Southwestern part,

1 but after the 15 minute break we won't hear from the middle
2 of the country, we will hear from the Southeastern part.
3 Let's take a 15 minute break.

4 [Brief recess.]

5 MR. LEWIS: Let's reconvene. I think we are going
6 to hear TVA's version of how it is in the trenches.

7 [Slides.]

8 MR. GLADNEY: I am Rod Gladney, with TVA. To give
9 you a little background, we installed Eagle -- we started
10 doing the planning in 1988, the later part of 1988 for
11 planning. At that time I was the engineer assigned the
12 responsibility of planning and implementation for Eagle 21.
13 I was out of our corporate maintenance organization at that
14 time. Right now I am working as tech support to electrical
15 and instrumentation supervisor at Sequoyah.

16 We backed up and looked at previous system
17 performance and plant performance to determine what we
18 needed to do upgrade-wise. There were several items that
19 were important. One was, we wanted to eliminate the bypass
20 manifold on the reactor coolant system to reduce radiation
21 exposure to personnel in containment during outages. That
22 could not have been done with the Foxboro hardware we had
23 because there was no way to do the signal combinations for
24 RTD's that was required for bypass manifold elimination.
25 Our system was maxed out. There was no capability to add

1 new modules to Foxboro hardware plus, there were no modules
2 to be had to add to the system if we wanted to.

3 AG analog hardware.

4 MR. CARROLL: I am surprised that Sequoyah had
5 Foxboro. Didn't you buy Sequoyah after Westinghouse stated
6 offering Hagen?

7 MR. GLADNEY: The reason that Sequoyah had
8 Foxboro, I can't give you detail. Both Sequoyah and Watts
9 Barr were designed with Foxboro.

10 AG analog hardware, it goes without saying, we are
11 talking hardware that was designed in the mid-1960's. Most
12 of the modules initial design was 1962 timeframe. The
13 modules were performing fairly well. We were operating with
14 tech specs that had a functional test frequency of quarterly
15 for everything except the Delta T channels. That frequency
16 was based on performance of the hardware, not drifting
17 substantially.

18 Significant manpower needed to calibrate and
19 maintain the system. Analog system, there is a lot of
20 components. Modules take a lot of time to do calibration.
21 To give you a feel for what we saw during refueling outages,
22 we spent roughly a week of three to five craftsmen time to
23 calibrate one Delta T channel, when you got through all the
24 modules involved in the channel.

25 Analog system calibration drift. I specifically

1 mention narrow range temperature here. The reason I mention
2 is, in analog space that channel had an electronics gain of
3 500. Any slight temperature variation in the case of the
4 module showed up in calibration.

5 Improved accuracy with digital systems. There were
6 fewer analog components involved. Once you do the analog to
7 digital conversion the accuracy is no longer an issue until
8 you go back to analog. Protection system logic upgrade. We
9 looked at how the plant had functioned since initial start
10 up and during some of our feedwater studies we identified
11 roughly 18 trips had occurred during start up evolution on
12 each unit, that we felt we could have eliminated with going
13 to new upgrade on the logic. Basically, the steam flow
14 functions were the ones causing us the problems.

15 Implementation of trip reduction measures. That
16 really goes hand in hand, steam flow items again. We also
17 wanted to look at some ways to reduce vulnerability during
18 start up on steam generator level.

19 Capability for automated surveillance testing. We
20 saw this as a big plus to reduce manpower and increase the
21 reliability and accuracy of our calibration evolutions. The
22 fewer hands on man hours and things involved the less the
23 chances of a personnel error or hardware that is out of
24 service being needed for plant use.

25 Capability to bypass the channel for testing.

1 This is something that Sequoyah presently hasn't implemented
2 but we have looked at it, and are seriously considering
3 pursuing it. It would allow us to keep all channels in a
4 normal mode, and the one that's in surveillance to be
5 bypassed instead of tripped like we presently do it. That
6 would reduce the unit vulnerability to a single failure
7 occurring during a surveillance test.

8 Simplified implementation of Reg Guide 1.97. At
9 the time we were talking about installing Eagle we were also
10 looking at fulfilling our commitments on Reg Guide 1.97 that
11 required additional analog isolation to be put into the
12 Foxboro cabinets, to separate the computer and control board
13 indicators. We did not have the physical space in the
14 cabinets to add that kind of isolation, nor could we buy
15 Foxboro equivalent hardware. So, we were facing a
16 substantial design change to modify the plant in some way to
17 add that isolation.

18 Flexibility for transmitter upgrades. The
19 Sequoyah design was a ten to 50 milliamp plant for
20 transmitter current signals. Most new transmitters now are
21 four to 20. This gave us additional flexibility to do that
22 change on a transmitter by transmitter basis if we wanted
23 to.

24 [Slides.]

25 MR. GLADNEY: Considerations that we looked at for

1 doing a plant upgrade were design, maintenance, procedures,
2 operations, and don't forget training. That's obviously an
3 important item to prevent plant problems in the future.

4 Under design considerations, things we looked at
5 in the initial process, obviously, drawing changes. A large
6 number of drawings get changed by this type of upgrade.
7 Functional requirements document revision to define how the
8 protection systems are supposed to function. Precautions,
9 limitations and set points document, basically scaling
10 accuracy document. Setpoint methodology -- excuse me,
11 that's the scaling accuracy and potential FSAR revision.

12 We also looked at tech spec revisions,
13 environmental qualification program revision, USQ associated
14 with new failure modes if there were any. We ultimately
15 filed the installation of Eagle as a USQ and went through
16 the resolution of that process.

17 Site acceptance test scope.

18 MR. CARROLL: You say you ultimately did that.
19 Was that initially your conclusion?

20 MR. GLADNEY: When we went through this initial
21 review we made that conclusion based on the history of where
22 the industry was on digital systems, the fact that we were
23 installing a system that had not been installed in any other
24 utility at the time, and it was a learning curve to go
25 through and assure that there is no software failures that

1 could impact safety, et cetera.

2 All these items were rolled into our decision to
3 say it's a USQ for Sequoyah's installation.

4 MR. SHEWMON: Sir, you talk about instrument
5 failures there, was the first time that I heard anything
6 close to a comment on software which is what we have heard
7 about here. Is this something where you bought yourself a
8 box and if it met specs somebody else had done all the
9 software verification that needed to be done, or what?

10 MR. GLADNEY: This is the first system of its kind
11 that we installed that had this kind of a software base.
12 When we did the initial look at it all the V&V for software
13 had not been completed. That is one of the items that
14 helped drive us to the conclusion that there was a potential
15 for software problems. Until we finished that V&V process
16 we had to keep that in mind toward the potential USQ
17 associated with the system.

18 MR. SHEWMON: If I am patient, I will hear more
19 about that or that was kind of why I didn't see it anyplace?

20 MR. GLADNEY: I really didn't go into the software
21 part of the system a whole lot. In Sequoyah's case the
22 software was written by Westinghouse and V&V by
23 Westinghouse, with TVA participation in looking at the
24 functional requirements for the software and NRC audits of
25 the V&V process from Westinghouse.

1 MR. CARROLL: Westinghouse did submit their whole
2 program to the NRC.

3 MR. GLADNEY: Yes. In fact, that V&V was
4 submitted before we had an SER to start the plant up.

5 MR. LEWIS: My internal quest to be quantified,
6 how big a software program are we talking about in any
7 terms, pages, lines, it doesn't matter.

8 MR. GLADNEY: It's a pretty big software package
9 in this case, because software is more than just protection
10 functions. It's also testing functions, the surveillance
11 testing. It's stored on four E-prongs on the processor
12 board.

13 MR. LEWIS: E-prongs come in different sizes.

14 MR. GLADNEY: Yes. I will have to give you a
15 number. I can't quote that number for you.

16 MR. LEWIS: Bits, lines, nothing.

17 MR. GLADNEY: It's a substantial number of lines
18 of software.

19 MR. LEWIS: How many?

20 MR. GLADNEY: A substantial number. It is not a
21 small program.

22 MR. LEWIS: Substantial doesn't help me a great
23 deal.

24 MR. KEMMERER: He was giving you a measurement in
25 inches, so that gives you an order of magnitude.

1 MR. LEWIS: Inches, at least use the metric
2 system.

3 MR. GLADNEY: That documentation should exist in
4 the V&V information.

5 MR. LEWIS: It was written in V&V by Westinghouse
6 and submitted to NRC. Did NRC read each line of code?

7 MR. GLADNEY: NRC actually audited the V&V
8 process, looked at all the discrepancies that V&V performers
9 solved during that process, and evaluated the resolution of
10 those discrepancies -- looked at the resolution of them.

11 MR. LEWIS: That's not the same as reading every
12 line of code.

13 MR. GLADNEY: No, it's not.

14 MR. LEWIS: Who read every line of code twice,
15 Westinghouse?

16 MR. GLADNEY: Westinghouse did. It was an
17 independent team, completely independent from the design
18 team, to do the V&V.

19 MR. LEWIS: There have been lots of experiments to
20 show that that doesn't do much good because people make the
21 same mistakes, of course.

22 MR. GLADNEY: Beyond that, there was a functional
23 test that software performers intended function --

24 MR. LEWIS: That was normal operating conditions.

25 MR. GLADNEY: No. That was on the manufacturing -

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

MR. LEWIS: Let's not go into more detail here. It would be nice to find out how big a system we are talking about, in any unit. I will buy any units.

MR. AHERN: There is approximately 1,200 software units of code, individual software units, combined to make up the total unit.

MR. LEWIS: How big is each unit?

MR. AHERN: I heard a number of around 10,000 lines of code for the total system.

MR. LEWIS: Ten thousand lines for the total system, okay. This is lines of source code in some language.

MR. AHERN: Yes, PLM 86 is the language that was used.

MR. LEWIS: That's not much.

MR. SHEWMON: It sounds like a lot to me.

[Slides.]

MR. GLADNEY: Site acceptance test scope. We had to evaluate what kind of detail was necessary to test the system to after installation, before we could declare it operable. Software verification and validation plan which we pretty much talked to you about here. Assessment of plant grounding.

MR. LEWIS: Let me continue the previous

1 conversation long enough to finish it. By the numbers that
2 Rich gave us earlier in the day which were 750 lines per man
3 month, 10,000 lines is one man year or, as Fred Brooks would
4 say, you could translate that into 12 man months. MR.

5 WYLIE: Did you change your grounding system?

6 MR. GLADNEY: No. That comment came up earlier
7 about grounding, and I wanted to try and address that here.
8 We assessed whether or not we had a grounding problem with
9 the system. We had Westinghouse come in and look at our
10 grounding system and even had an independent contractor come
11 in and look at the grounding system.

12 We did not identify a failure in the grounding
13 system that had caused problems in the analog system. Our
14 basic conclusion was that we did not have a grounding issue
15 that needed to be addressed for installation of this system.
16 We also, after installation, had power line analyzers
17 monitoring the AC line to the system for several months of
18 time and never saw any perturbations show up that would have
19 come out of either the ground spike or AC spike coming into
20 the system.

21 MR. WYLIE: Is that a separate instrumentation
22 ground?

23 MR. GLADNEY: No, it is not.

24 MR. KEMMERER: Excuse me. Is this a control
25 system?

1 MR. GLADNEY: No, it's not.

2 MR. KEMMERER: Just safety --

3 MR. GLADNEY: Reactor protection, just safety.

4 MR. KEMMERER: But you have it testing in with
5 safety.

6 MR. GLADNEY: Yes. The outputs of this system do
7 drive control channels. It's basically -- this is a reactor
8 protection piece of the Westinghouse instrument package for
9 their plants. So, the protection channels drive control
10 channels to protect their function. Of course, we had to
11 meet the criteria for control and protection interaction in
12 that design.

13 MR. WYLIE: When was this installed?

14 MR. GLADNEY: The installation began in March of
15 1990, and was completed in about 40 days of time.

16 [Slides.]

17 MR. GLADNEY: Maintenance considerations. Reduced
18 time performance surveillance tests. Reduced times to
19 perform calibration of dynamic functions which is
20 substantial in an analog system. Calibration of a dynamic
21 function requires quite a few pieces of test equipment and
22 time to sit there and generate the ramp rates necessary for
23 calibration and then monitor the output and if it's not
24 accurate, tweak the adjustments and put a new ramp back in
25 and see how it comes out in calibration that time.

1 Capability for testing and bypass, we talked about
2 that a little. It's not implemented for us at this point.
3 Potential scaling problems with site specific programs.
4 That's kind of a vague term. What we are really looking at
5 is, was there any site specific program that changing our
6 hardware would cause a scaling problem, some unique
7 calibration program we were involved in that hadn't been
8 considered in the original system design.

9 Tighter tech spec allowable values. One of the
10 decisions made at the plant was, we did not want to move set
11 points. We wanted to keep plant set points at the values
12 everybody knew. The basis for the difference between the
13 desired setpoint and allowable value is drift of the
14 instrumentation. In this case digital system has a smaller
15 drift term, so we ended up moving our allowable values.

16 Potential impact to response time testing program.
17 Of course, we do response time tests on the analog channels.
18 We recognize that this would cause a perturbation to all the
19 test procedures we had on the analog system which also
20 included response time testing. In some cases we had to
21 back up and rethink how our tests were structured, to ensure
22 response time verification.

23 New failure modes to diagnose and repair. We have
24 talked a little on that several times today. Analog system
25 is pretty much a box concept. You can walk through

1 different boxes and find which box is bad. In a digital
2 system it may not necessarily be that clean cut, so it's a
3 retraining evolution for your maintenance people to make
4 sure they are aware of how to troubleshoot and diagnose the
5 problems in these systems.

6 MR. CARROLL: Bottom line on maintenance, how many
7 have you been able to effect a change in the number of I&C
8 maintenance.

9 MR. GLADNEY: We have greatly reduced the I&C
10 workload for calibration and functional tests on this
11 system.

12 MR. CARROLL: What does that translate into, in
13 bodies.

14 MR. GLADNEY: In bodies, we are now using those
15 bodies to do other jobs that help improve plant reliability.
16 We did not reduce bodies. We kept the bodies in the areas
17 we wanted to improve.

18 In terms of man hours reduction, I talked roughly
19 a week of three to five people time for Delta T channel. In
20 this case we are talking six hours for that same channel.
21 It's a substantial reduction in man hours, testing.

22 Procedures considerations. Clearly, we had to
23 revise the functional test, the channel calibration
24 procedures, response time tests like we already talked
25 about, channel check procedures from the control room, and

1 calorimetric program. It was deeply ingrained in how the
2 original instrumentation functioned and test points in the
3 original system.

4 We also revised the RCS flow recalibration program
5 because our original design basis does not allow an
6 inaccuracy on the transmitter. We compensate for that
7 inaccuracy with a rack calibration during start up
8 procedures.

9 [Slides.]

10 MR. GLADNEY: RCS/RTD cross calibration program.
11 We revised how that test was done. There has been a lot of
12 discussions on that test. Right now, we use a data logger
13 to require data from the protection systems and reduce that
14 data, to determine if we have an RTD out of calibration. If
15 so, we tweak the calibration coefficients in this system to
16 compensate for any error on the RTD.

17 Program to remove failed channels from service.
18 To comply with tech specs on the failure we developed a
19 special procedure that laid out all the details necessary
20 for every instrument channel, to remove that channel from
21 service, replace the bistables in trip within the time
22 allotted by tech specs. Clearly, that procedure had to have
23 a big overhaul because of the new type hardware.

24 Annunciator response procedures. We maintain a
25 procedure that tells the operator how to respond for every

1 major annunciator in the control room. Those responses
2 changed, partially because there are new annunciators added
3 with this system to help the operator diagnose failures in
4 the system.

5 Procedure to monitor and update RCS streaming
6 coefficients. Streaming was a new factor for us, that we
7 got into when we eliminated bypass manifold. We had to add
8 that to our testing program.

9 Under operations, some of the same items will
10 appear again. New annunciators for protection system
11 failures to help him understand the magnitude of that
12 failure if it occurs, so that he will know what type of
13 plant actions to take. New protection logic to learn. We
14 eliminated steam flow and went to Westinghouse's new steam
15 break protection which changed our logic substantially.

16 MR. SHEWMON: I don't know what a reactor
17 protection system is. Could you help me out. This is a
18 safety system?

19 MR. GLADNEY: Yes, this is a safety system. The
20 signals from the process transmitters measuring for instance
21 pressurizer pressure and level, reactor coolant flow,
22 reactor coolant temperature, come to this system. This
23 system massages them -- for lack of a better word -- and
24 generates trips when that parameter reaches some predefined
25 value, some set point.

1 MR. SHEWMON: This is enlightening in one regard,
2 in that this theory of having a simple program which only
3 had a few inputs in the scram circuit is 12,000 lines which
4 may be short by some people's standards.

5 MR. GLADNEY: I will give you a little feel for
6 it.

7 MR. GALLAGHER: I think Mr. Ahern said there were
8 1,200 software units. Each one would have -- I think that
9 the lines of code would be on the order of around 15 and 20-
10 some thousand lines.

11 MR. SHEWMON: That's within 20 percent of his off
12 the cuff. I will take that as good enough for now.

13 MR. GLADNEY: The magnitude of system that we are
14 talking about here is 13 cabinets of instrumentation. We
15 replaced the entire protection system at Sequoyah. Those 13
16 cabinets are divided into four channels of instrumentation.
17 Within each channel you have basically one of the redundant
18 sets of instrumentation. Some functions are in three
19 channels and some are in four.

20 MR. SHEWMON: You have answered my question for
21 now, thank you.

22 MR. KEMMERER: Was each channel running on a
23 separate processor?

24 MR. GLADNEY: Each rack is running on a separate
25 processor, and there are up to four racks in a channel.

1 Each channel is divided among processors, anywhere from two
2 to four processors.

3 MR. CARROLL: One of the things that ought to be
4 said to Paul's comment is that a lot of these lines of code
5 are devoted to surveillance or automated surveillance and
6 testing, checking and so forth, and not to the basic
7 protection.

8 MR. GLADNEY: And, that adds a lot of overhead to
9 a system. There is a lot of additional programming that was
10 done specifically to enhance surveillance testing and
11 minimize the time we will spend during that testing.

12 MR. COBB: Yes, but all of that has to be included
13 in the verification too, because surveillance testing is a
14 double edge sword there. You get benefits from that, but
15 you also get potential failure situations from that. That's
16 the reason the verification process that I tried to talk
17 about this morning are so important and could not be
18 overlooked.

19 MR. CARROLL: I agree with that completely. I was
20 just trying to put it in perspective for Paul. Protection
21 functions are always --

22 MR. SHEWMON: There are some virtues to having a
23 simple metallurgist around but there's a certain overhead
24 that goes with it.

25 [Laughter.]

1 MR. COBB: If you look at any software system, the
2 functioning that you think that it's going to be doing is
3 always the minor part of the total lines of code. All of
4 the things that take care of the things that can go wrong
5 are the major part of the code.

6 People get in trouble if they only develop the
7 mainline stuff, and then add all the other stuff on as
8 second thoughts.

9 MR. LEWIS: But he's not a simple metallurgist.
10 [Slides.]

11 MR. GLADNEY: We briefly mentioned earlier a new
12 tech spec action statements. Also, we significantly reduce
13 risk of reactor trip from steam generator level transients
14 during startup and during low power operations. This is
15 some more of the functional upgrades we implemented that
16 added two new programs to us for environmental allowance,
17 modifier and trip time delay. Those programs effectively
18 allow us more operating margin on steam generator level on
19 start up at the low power when you are subject to
20 shrink/swell problems.

21 Training considerations. Clearly, new steam break
22 protection, environmental allowance modifier and trip time
23 delay functions, explaining how those circuits function.
24 Make operations clearly aware of them. Median signal
25 select. There's that term again, on steam generator level

1 control.

2 In our case median signal select is totally an
3 automatic function. It has a module installed in our
4 control racks that monitors all three protection signals
5 from the level transmitters and selects the signal that is
6 the middle of the three to control steam generator level.

7 Training considerations. Clearly, training of the
8 instrument technicians on the hardware, so that we
9 understand how it functions, on how to trouble shoot the
10 hardware, on how to handle the circuit boards for the
11 system, how to perform the new surveillance procedures, how
12 to do parameter updating based on site specific data.
13 Parameter updating is basically the old version of it. We
14 replaced for calibration with a parameter update that you do
15 from a keyboard.

16 Use of user configurable test points and test
17 carts. One problem we had seen in the previous systems was,
18 when a problem occurred in the plant and you wanted to go
19 out and diagnose it, you spent a lot of resources hooking up
20 test equipment to take data. In this case we can go down
21 and plug in a test cart and program test points that are
22 completely removed from a protection panel, to monitor that
23 function and provide it to us on outputs so we can hook up a
24 recorder. Or, you can read it directly off the test cart,
25 too.

1 Lessons learned from our implementation. Let me
2 back up a second before I go on. To give you a feel for --
3 this is what we did have and this is what we have now.
4 This was Foxboro H line. Everything was individual modules.
5 You had a door that flipped up to inject your signals in.
6 What we have now, the picture doesn't show it very well from
7 here.

8 In this is microprocessor that runs the cabinet.
9 It is subdivided. There is a microprocessor sitting here
10 that does all the calculations. There's a board right beside
11 it that does the analog to digital conversion. All the
12 analog inputs are on these horizontal boards right here and
13 behind this panel. Analog outputs and bistable outputs are
14 right here.

15 It's functionally a system that takes analog
16 signals in from the transmitters, does some signal level
17 conversion to get the signal to the right voltages to do an
18 analog to digital conversion on it, does that conversion.
19 From that point on it processes everything in digital
20 numbers until you get to the final output for analog, and
21 then it converts it back to analog.

22 For trip functions it sends a signal out every 100
23 milliseconds to the bistables, to tell them to stay
24 energized or to de-energize. If we have a hardware failure
25 where the software fails to function or a processor board

1 dies for some unknown reason, that update stops occurring
2 and the bistables go trip within about 150 milliseconds.
3 It's sitting there monitoring itself, expecting to get an
4 update. If it doesn't get that update it trips, no matter
5 what.

6 MR. LEWIS: You said something about digital
7 analog board.

8 MR. GLADNEY: Yes.

9 MR. LEWIS: How many bits wide is the digital
10 signal?

11 MR. GLADNEY: I think it's a 16 bit conversion.

12 MR. LEWIS: You can get single chips that do 80
13 conversion on 16 bits. I am just wondering what the board
14 does.

15 MR. GLADNEY: This is an industry standard board
16 that has a multiplexer front end to take signals from a lot
17 of analog channels, and sequentially digitize the signal.

18 MR. LEWIS: Most of that is multiplexing,
19 probably.

20 MR. GLADNEY: Right. It has its own processor
21 sitting there to do all this calculation, this conversion
22 for you.

23 MR. LEWIS: The DA conversion is a matter of one
24 chip.

25 MR. GLADNEY: The conversion by itself is one

1 chip, and you have the multiplexing front end on it.

2 [Slides.]

3 MR. GLADNEY: On the center of the front is a test
4 panel. This is where we plug in our test cart. It has a 25
5 pin connector the test cart plugs in through, to allow us to
6 monitor rack parameters or to do surveillance testing.

7 Lessons learned from implementation. I will
8 clarify how I built this slide. Going into the outage we
9 had a certain conceived notion on system configuration and
10 everything we expected to happen as we installed it and
11 start the plant up. We learned some things in the process
12 that we have given back to Westinghouse to help other
13 utilities apply it to future installations.

14 Indicated full power delta T changed. I say
15 indicated, and that's very important. That is based not on
16 the fact that we put it in Eagle, but the fact that we
17 changed how we monitor temperature in the hot legs. The old
18 scoop tube combination that sent that signal to a bypass
19 manifold and measured temperature in the bypass manifold is
20 no longer applicable. We now have three RTD's in the pipe,
21 the points where they exist in our pipe, since there's a
22 hotter temperature than the bulk water temperature.

23 Two items lead us into that scenario. One is
24 streaming the way the hot water comes out of the core going
25 into the hot leg piping, it's not a mixed temperature water.

1 There is stratified sections in that water, the low leakage
2 core which tends to concentrate heat in certain sections of
3 the core, both add to that phenomena.

4 Revised methods to compensate for nuclear
5 instrumentation system gain. Our original program, we went
6 down to an analog module and tweaked some calibration
7 adjustments, once we knew what NIS gains were so that we met
8 the right curves on delta T calibration. That no longer
9 applies. It's now done as some coefficients that are
10 updated into the system.

11 Requirement to periodically measure streaming
12 coefficients and update Eagle 21. Streaming issue again.
13 To ensure that the stratification in the water hot leg does
14 not change and adversely affect temperature, we periodically
15 do this. We are doing it on a monthly basis to make sure -
16 -quarterly basis during our functional test, to make sure
17 it's not a problem.

18 MR. CARROLL: How do you do that?

19 MR. GLADNEY: Some more overhead and software.
20 There's a routine running all the time that looks at
21 streaming on all three RTD's and calculates any errors
22 associated in streaming. It actually prints out to us,
23 streaming offset to enter back in. It does not put it back
24 in it only tells us, and we have to manually put it back in.

25 The requirement to periodically measure RTD

1 current and update system. Eagle uses a true four wire
2 signal to the RTD where it's a current source going to it
3 and a voltage sense coming back. This way, we ensure that
4 no drift in the current source would cause a shift in
5 calibration to the Delta T channel. If that current changed
6 what it would effectively look like is a temperature change
7 to you. We make sure that the current is not changing and
8 that our temperature calibrations are accurate.

9 Environmental allowance monitor cannot be operable
10 during containment vent or purge on ice condenser plants.
11 That's an item that wasn't clearly discussed up front. We
12 understand the driving force is the way the analysis is done
13 for harsh environment in containment. We disable that
14 function if we are doing a vent or purge operation.

15 MR. CARROLL: I am not sure I understand what an
16 environmental allowance monitor is.

17 MR. GLADNEY: It monitors the steam generator
18 level channels -- it monitors containment. If containment
19 pressure changes, then we take our some margin allowed in
20 steam generator level calibration and make the trips occur
21 at tighter values. If containment temperature level goes up
22 our level calibration degrades.

23 We must go to 100 percent power to initially set
24 streaming coefficients. That's so that we can be sure that
25 our streaming values are accurate for full power operation.

1 There were some initial values we could use in the system
2 until we hit 100 percent power.

3 Must remove stillaming coefficients on Delta T
4 channels to perform functional tests and calibrations. This
5 is an item that we felt so strongly about that we worked
6 with Westinghouse and have revised the software and that no
7 longer applies, and won't apply to any future utility.

8 [Slides.]

9 MR. GLADNEY: No ability to quickly download or
10 upload tuning parameters into our out of Eagle. The system
11 is microprocessor based. There is no access into or out of
12 that system onto disks. It is all controlled on E-prong.
13 There are some tuning parameters, software screwdriver, to
14 set bistable setpoints and gain coefficients for various
15 modules. Those are the parameters that I am referring to.

16 As it is right now if we have a hardware failure
17 in the system and end up having to replace the components
18 that store those values, we have to manually re-input the
19 parameters. That's what we are going after. At some point
20 in the future we would like to have some way to do that, do
21 a full verification they are correct, and be able to get the
22 system back on line in a quicker fashion than manually
23 entering the values.

24 This is a new item that wasn't possible prior to
25 this system. Different racks independently monitor for the

1 same transmitter. The trip time delay, we talked about
2 earlier, is performed in our case rack one for protection
3 set one. The Delta T function is performed in rack two.
4 Both functions require monitoring reactor coolant system
5 Delta T because trip time delay looks at Delta T to
6 determine when trip time delay is allowed and when it is not
7 allowed.

8 If a Delta T rack dies our trip time delay keeps
9 functioning because both racks independently perform the
10 conversions of the voltages from the RTD to temperature.
11 Either rack failing does not inop the other rack.

12 There was some penalty associated with doing it
13 that way though. It meant now, when we had to update an RTD
14 coefficient for instance because of the cross calibration,
15 we have to make that change in two places. That's an
16 acceptable penalty to minimize hardware failures from
17 impacting the unit, though. We also talked about RTD
18 current changes. If that current changes we have to update
19 two racks on that current change.

20 Initiated trip from a partial trip switch on
21 energize to trip functions if the software bypass has been
22 established. As you are aware, there are very few energize
23 to trips because that's normally not the mode you want to
24 operate. There are a couple of places that it's prudent to
25 operate that way. This was something that we just didn't

1 understand up front. It is not a safety implication in
2 operating it the way the system operates, and we just had to
3 put our procedures in line with how the system functions.

4 We cannot retrieve error messages after reset of a
5 rack. It's a microprocessor. It stores errors when errors
6 occur. We can go down and read those errors prior to
7 initiating a reset of the system. Once we have reset it,
8 that error no longer shows in the system.

9 Failure of a loop processor that causes digital
10 outputs to trip and analog outputs to freeze where they are.
11 That's a choice we made. We stand by that choice because I
12 want the protection functions to go to trip but if the rack
13 failed I don't want to put the unit in a transient because
14 of a single hardware failure. I want the operator to take
15 the couple of minutes it takes him to recognize the trips he
16 got on the board and diagnose that that means I had a rack
17 failure, and then transfer control where he needs to.

18 Any other process sets you up for unit transient
19 that the operator is trying to catch instead of trying to
20 head off.

21 Racks draw more current than the original analog
22 hardware. That was a surprise to us, and it's a surprise to
23 several other utilities when they first started looking at
24 the system. The only value in it here is, utilities need to
25 understand that they may not have the power capacity in

1 their present system design if they don't look at it up
2 front.

3 Connection of maintenance and test equipment to
4 test points can cause negative spikes and momentarily
5 actuate bistables. That was a lesson learned to us. When
6 we go back into the analog in this system the capacitants of
7 the digital volt meters were used, can cause a momentary
8 spike if you plug it into a test point. We have taken that
9 into account in our procedures and basically we don't use
10 volt meters on those analog inputs unless there is some
11 trouble shooting evolution we are going through, with
12 operations fully aware of what's happening.

13 [Slides.]

14 MR. GLADNEY: The benefits of the system. There
15 are quite a few that we have seen. Significant reduced
16 calibration drift, reduced functional test calibration,
17 significantly reduced loop calibration time. The functional
18 test was a relatively short test for our old system. It got
19 shortened a little bit, but most of the time for that
20 functional test is administrative time and not hardware
21 time. You are trying to do coordination through our control
22 room, make sure there aren't any other plant parameters in a
23 condition that would cause an adverse plant reaction if we
24 took a channel out of service.

25 This type of administrative process takes as much

1 time if not more than the physical performance of the
2 functional. That same administrative time on the
3 calibration -- the calibration is a much more detailed
4 process and requires a lot longer time.

5 Significantly reduced time to implement rescaling
6 of the instrument loop during testing after refueling
7 outages. RTD cross calibration is a prime example. If we
8 find a RTD calibration that is off and we have to go in and
9 correct it, in this system it's a matter of about 15 minutes
10 worth of work. In the old system it was a matter of
11 potentially recalibrating the entire analog stream to get
12 channel accuracies back in line.

13 Simplified special testing to eliminate
14 connections to each analog loop. We don't have connections
15 to the analog loop anymore because in most test evolutions
16 we can go right into test points that are electrically
17 isolated from the processing channel. We can get direct
18 printout of calibration data and any present process value
19 in the plant directly off the test equipment.

20 We eliminated the need for transmitter adjustments
21 at power on steam flow, feed flow and reactor coolant flow.
22 We can now do that compensation remotely from the auxiliary
23 instrument room, and it reduces dose to people and time to
24 actually perform the work.

25 Improved operator confidence in steam generator

1 level control with new setpoints and time delays provided by
2 ETM and TTD. If, for no other reason, that alone will save
3 us a large number of dollars and potential risk to the plant
4 by reducing trips that aren't needed on the plant. Like I
5 said earlier, our pre-installation analysis showed that
6 roughly 18 trips on each unit had been caused by steam flow,
7 feed flow -- the old steam logic that existed and not having
8 wider margin on steam generator level during initial start
9 up.

10 That, alone, has already saved us several trip
11 evolutions that we probably would have tripped from in the
12 old system. Reduced vulnerability to reactor trip based on
13 the steam break logic because of the wider operating
14 margins.

15 Are there any questions that I can answer?

16 MR. LEWIS: Could I, perhaps, just start with one.
17 Just thinking a little bit about security, why are you sure
18 that there is no trojan horse lurking in the software?

19 MR. GLADNEY: For several reasons. One, a very
20 extensive V&V process on the software. Two, an extensive
21 factory acceptance test to prove all functional requirements
22 were fulfilled by the software as written. Three, the start
23 up testing, where we actually did calibrations on all the
24 channels and verified that they performed accurately across
25 their calibration range.

1 MR. LEWIS: I am not a great expert, but I didn't
2 think that any of those things would detect a trojan horse.

3 MR. GLADNEY: The V&V process is your strong
4 supporter of detecting that trojan horse.

5 MR. PLACE: I think the V&V process would, if you
6 were looking for a trojan horse. The V&V process should
7 detect one, if you are looking for it.

8 MR. LEWIS: If you are looking for it.

9 MR. KEMMERER: I thought V&V was done by the same
10 person that wrote the code, or same company.

11 MR. GLADNEY: Same company, independently.

12 MR. KEMMERER: As opposed to an I V&V that might
13 be looking for trojan horses.

14 MR. GLADNEY: The group that did the V&V was
15 specifically chosen to ensure independence from the people
16 that wrote the software.

17 MR. KRESS: If I were mad at you and wanted to do
18 something to your system and I was the guy that did the
19 maintenance on your computer system, would you be vulnerable
20 to me if I was a pretty good hacker and wanted to put the
21 equivalent of a virus into this system that developed after
22 you started up?

23 MR. GLADNEY: You don't have the capability to put
24 that virus in the system.

25 MR. KRESS: There is no software that is

1 changeable other than changing out hardware itself.

2 MR. GLADNEY: The only control to change that
3 software is through the test cart that was built directly to
4 interface directly with the system, that allows you touch
5 screen access to the system only. You can't go in and type
6 in code to the system.

7 MR. KRESS: Can I type in stuff to the test cart?

8 MR. GLADNEY: No. You don't have access to a
9 programming level. You have a graphic user interface that
10 allows you to change parameters, and you can type in the
11 parameter number that you want to change to. That's the
12 extent of what change you can make.

13 You can't go in and change any operational codes
14 in the system to change how the processor functions.

15 MR. LEWIS: Why are you so sure of that? That's
16 the intended channel but of course most accesses of course
17 use unintended channels into the software. Do you really
18 know at the key and signal level, that the parameter
19 changing channel cannot be used for other purposes? Was
20 that a design intent?

21 MR. GLADNEY: The original intent was that that
22 test cart would be the only vehicle used to go in and access
23 the system. That cart talks only to the tester. It does
24 not talk to the calculation processor. That tester --

25 MR. LEWIS: Is physically separate?

1 MR. GLADNEY: Is physically separate. Once that
2 tester has accepted the data that you want to transmit and
3 concurs that the data is valid data for the processor to
4 accept, it will then transmit it to the processor. You have
5 no direct access to the calculation process.

6 MR. LEWIS: I understand that that's not the
7 intent. I don't have anything specific in mind, but I have
8 read enough stories to know that there are always unintended
9 back channels into a system.

10 MR. GLADNEY: Let me pose it a different way. If
11 that same person had that intent with an analog system
12 during surveillance testing, couldn't he have done the same
13 equivalent thing.

14 MR. LEWIS: That's the argument which, when I was
15 in college, we used to describe by saying yes, but it isn't
16 true in the South.

17 MR. GLADNEY: I think that possibility has always
18 been there in our plants, and we rely on our security
19 systems and the qualification of our people and our
20 background checks on people to help ensure us that no
21 saboteurs are among us.

22 The system is pretty self-protective of somebody
23 going in there and doing that. If they try and do it, the
24 operator is going to know that somebody is in that cabinet
25 from an alarm on the door itself. He will know if somebody

1 is into parameter update mode from another alarm that comes
2 in the cabinet.

3 If it's outside of the surveillance test he is
4 going to know what's going on. Coming out of that
5 surveillance test we have our channel checks and
6 verifications that redundant channels are matching.

7 MR. LEWIS: It sounds to me like a while back I
8 was at a DOE plant and asked a related question, and was
9 told that the telephone lines that go into the computer are
10 only used for output and never for input. It was a new kind
11 of telephone line that I had never heard of.

12 I think it pays to take these issues a little more
13 seriously than people do, and it pays to not get people like
14 us but sneaky people to look at the options because they are
15 the ones that know the game better than we do.

16 MR. GLADNEY: I fully agree with you. We have to
17 be sensitive to that type of issue, and not set the plants
18 up for a problem in that fashion by virtue of the fact that
19 the software was written to communicate directly to this one
20 specific test cart.

21 MR. LEWIS: I am just reacting to the fact that
22 most of your answers have been in the direction of
23 describing how the system is intended to work under normal
24 conditions. One really has to look at the next level, of
25 how it actually works to find what these sneak paths are.

1 There are just lots of scare stories along those lines.

2 MR. KEMMERER: Relative to your question about
3 would this happen with the analog system, with the analog
4 system you didn't have that testing capability in there;
5 isn't that correct? I mean, there wasn't a way of going in
6 and reprogramming the analog system even if it is controlled
7 reprogramming in that testing.

8 MR. GLADNEY: The technician really had access to
9 the same level of potential for making it not do a safety
10 function for an analog system as he does with digital. He
11 had full adjustability on bistable setpoints.

12 MR. CATTON: But there's no memory in the analog
13 system. I think that's what Hal was talking to.

14 MR. GLADNEY: Yes, there is.

15 MR. CATTON: There is?

16 MR. GLADNEY: In terms of a setpoint. It's a
17 fixed pot.

18 MR. CATTON: Yes, but the pot doesn't have any
19 capability of changing itself.

20 MR. GLADNEY: No. It would have to be in terms of
21 writing something that goes down the road and changing it.
22 That is different.

23 MR. CATTON: I think that's what these people were
24 referring to.

25 MR. KEMMERER: We also heard a while ago that

1 don't feel bad about the 12,000 lines of code of 15 to 20,
2 because a lot of that is the testing code and surveillance
3 code. That's probably the code that was looked at the least
4 in whoever was reviewing this.

5 MR. GLADNEY: It went through the exact same
6 process.

7 MR. KEMMERER: It went through the same process,
8 okay. But not checked for security -- presence of a trojan
9 horse or whatever. The big difference here is -- I am
10 actually surprised that this was brought up. The big
11 difference here is that you can go in and change what is
12 running on the system in a limited manner, just as in a
13 limited manner when you go up to an ATM. It's programmed to
14 do specific things.

15 But it may be that you can go up and get it to do
16 something different if it has a trap door or trojan horse in
17 there with the appropriate sequence. You are absolutely
18 right, that with your background checks and so forth, that
19 it requires a trojan horse written in there. It requires
20 someone getting the job at the company that happens to have
21 access to the cart and all those other things.

22 On the other hand, you don't want to ignore it
23 completely.

24 MR. GLADNEY: No. Clearly, also, the periodic
25 testing helps prove that that hasn't happened.

1 MR. KEMMERER: Actually, I don't believe that at
2 all. Periodic testing is going in and saying okay, when we
3 use this test cart the normal thing to do is check these
4 points here. It goes and checks those points. It never
5 tries the sequence that is set up that says if you see this
6 sequence on the test cart do automatic shutdown.

7 I mean, you are worried about shutting off safety
8 functions and maybe what I am trying to do as a terrorist is
9 actually to shut things down.

10 MR. GLADNEY: The way the testing works here is,
11 the processor doesn't know it's being tested. It has no
12 access to the information that says it is in test. A signal
13 is injected by this test subsystem to go out there and check
14 the processor.

15 MR. KEMMERER: The test subsystem shares the same
16 processor that --

17 MR. GLADNEY: No, sir. Two, independent
18 processors.

19 MR. CATTON: That helps.

20 MR. KEMMERER: I am confused as to the number of
21 processors. You said there is per bank there's two to four
22 processors.

23 MR. GLADNEY: There are two to four racks in each
24 protection set, each channel. Inside every rack there are
25 two independent processors, one doing the protection

1 functions and one sitting there running continuous
2 diagnostics.

3 The one running diagnostics is call the tester
4 processor. It is the one that goes out and injects
5 surveillance signals for testing. It is totally independent
6 from --

7 MR. KEMMERER: What does the term inject
8 surveillance signals mean.

9 MR. GLADNEY: It actually takes the signal and
10 converts it to analog, and injects it back in the front end
11 of the analog board for the system.

12 MR. KEMMERER: That analog board goes to both
13 processors, right?

14 MR. GLADNEY: No. Then, the calculation processor
15 looks at the digital data that the analog injected signal
16 was converted to, massages that as if it was a true process
17 input and then tells the tester I just calculated this
18 value.

19 MR. LEWIS: You have to be extremely careful about
20 using these anthropomorphic words like something tells
21 something else and somebody injects something into them. In
22 every case there's a wire connecting them. When you use a
23 term inject it means you have the idea of the wire carrying
24 a signal in one direction. Any wire can carry signals in
25 two directions. It depends on what is at the ends of the

1 wires that determines that.

2 As far as periodic testing the reason I picked a
3 trojan horse was to think of something that would test out
4 just fine except on July 31, 1998 or something like that.
5 You never discover that in periodic testing until July 31st.
6 It was the famous Michelangelo which was nothing at all, as
7 it turned out. The concept has often been used; things that
8 work just fine except in a particular time.

9 There are a zillion legitimate computer failures
10 that occur on leap year. It happens every four years, and
11 things like that. You can't use these anthropomorphic terms
12 about testing and be confident about what's happening.

13 MR. GLADNEY: One of the things that makes
14 Michelangelo style virus very devastating is, most computer
15 systems are set up with disk access where that disk can
16 carry that virus in. That is a level that is not accessible
17 here. You can't do that. It forces somebody to really want
18 to be a saboteur before you are going to get that happening
19 in a system like this.

20 MR. LEWIS: I recommend that you be a little less
21 confident. I also recommend that everybody read the Academy
22 report on this subject, which was a rather good report. It
23 contains in it a rather good virus written by -- I have
24 forgotten his name. He wanted to demonstrate that he could
25 write a virus that nobody would detect, and then he gave the

1 code for it in the report.

2 It is just awfully easy to err on either side,
3 being an alarmist or being confident. I think it pays to be
4 somewhere in between.

5 MR. CARROLL: Just as a general comment, you were
6 concerned about people wanting to shut the plant down.
7 There are a lot easier ways to do it than getting inside the
8 plant and doing something with software.

9 MR. GLADNEY: Yes.

10 MR. CATTON: I think in this case you wouldn't
11 have to get into the plant. You would have to get in the
12 place that makes the Roms for the plant.

13 MR. CARROLL: All I am saying is that putting
14 three sticks of dynamite on four legged trans^{mission} tower
15 connecting the plant to the grid, is pretty simple.

16 MR. KEMMERER: Pick what you want to do. I can be
17 pretty safe. I can already be in Brazil or wherever I am,
18 after I program this thing as Hal says, to go off at some
19 particular time or given the particular sequence or whatever
20 it is.

21 MR. CARROLL: My only point is, the only thing I
22 am concerned about in terms of sabotage is simply sabotaging
23 it so that it does not perform its function.

24 MR. KEMMERER: The reason I said shutdown in the
25 question which I am still not sure what the answer is, is

1 there anything that the testing cart can do in terms of
2 putting in or the process of the tester that it talks to,
3 nothing from there ever goes to the normal processor; is
4 that what you are telling me.

5 MR. GLADNEY: No.

6 MR. KEMMERER: It's absolutely isolated.

7 MR. GLADNEY: That port only communicates to the
8 tester, not to the calculation processor.

9 MR. KEMMERER: I don't understand what it is
10 testing then. I am sorry, I am slow.

11 MR. GLADNEY: The code is written inside the
12 system that takes all the outputs for protection functions
13 and transmits them to this tester through one communication
14 link, through an isolated communication link. You are
15 right, bidirectional communication could occur.

16 The tester then takes that data and assesses
17 whether or not the outputs justified the inputs that it sent
18 to the system or matched the inputs. If they do not match
19 it comes back and says this doesn't match. Here's a
20 printout of the deviations I saw.

21 MR. COBB: The tester is testing at the
22 microprocessor?

23 MR. GLADNEY: Yes, it is.

24 MR. COBB: The only way you are going to answer
25 the questions that you are raising is to look at the black

1 box functions that all of these things purport to do, and
2 see exactly what they do and then have a proof that the only
3 thing that exists inside that computer is exactly what is in
4 that black box function. No more and no less, and exactly
5 that.

6 Until you can -- you can talk in any level of
7 detail you want about these systems. But until you actually
8 know what the mathematical function is written in terms of -
9 - you don't have to think of A plus B plus C -- written in
10 the terms of what the language of what this system is and
11 the stimulus and responses, after you have that black
12 function and until you see exactly what it does - because
13 the beauty part of a black box function is that it is only
14 written in terms of things that are eternal to it. You
15 don't have to worry about a memory because you know what it
16 is, and then you have to have the software that is
17 functioning inside of there only executes that function.

18 Then, you have to have some way to keep it under
19 control. The question you are asking about, keeping all of
20 this stuff under control after it's developed, it's a
21 security process which I think people -- I assume that
22 people know how to do something -- I don't know how to do
23 that.

24 MR. KEMMERER: The question isn't keeping it
25 secure after it's developed, it's keeping it secure when it

1 is developed. I think that's what Hal was getting at.

2 MR. COBB: I understand what to do during
3 development and what that proof is. After I have that proof
4 and all done, then I understand we have to keep it secure. I
5 believe I have some understanding about how to keep it
6 secure during development. I assume that other people know
7 how to keep it secure after development.

8 That's something I don't know how to do. I am not
9 a Brinks Armored Car person and things like that.

10 MR. GLADNEY: Maybe this is an area that through
11 the V&V process or some guidance issue we can strengthen the
12 V&V process, to make sure people consider these areas of
13 concern.

14 MR. COBB: That's one of the comments that was
15 sort of periphery to the person who spoke just before lunch,
16 was talking about some of these things. That is all part of
17 the process. I agree.

18 MR. KEMMERER: Your testing is completely passive,
19 as far as the system goes, the protection system.

20 MR. GLADNEY: Yes.

21 MR. KEMMERER: Which is running on its processor.
22 All it is doing is feeding out values that your tester then
23 says now I am going to sample some of these.

24 MR. GLADNEY: Right.

25 MR. KEMMERER: When you talk about running a test,

1 it's just taking the values that are always there and
2 sometimes looking at them and other times not looking at
3 them. It is not -- what I mean between passive and active
4 is, active means I will input something to go and see how
5 the processor works. It's only passive.

6 MR. GLADNEY: No, it is not. It is active.

7 MR. KEMMERER: If it's active, then you must be
8 doing something to the other to the other processor, or else
9 to the hardware that the processor is looking at.

10 MR. GLADNEY: It's the hardware.

11 MR. KEMMERER: If you are doing something to the
12 hardware and that processor is looking at it, then it gets
13 to the processor -- the transitivity.

14 MR. GLADNEY: Let me try and explain how it does
15 it. There is relay on the front end of the analog board
16 that switches between the field signal and the test signal.
17 That relay is only actuated through the test subsystem. The
18 processor doesn't know whether that relay is actuated or
19 not.

20 MR. KEMMERER: When it is actuated and you get
21 input from that system, that input is going into the actual
22 processor --

23 MR. GLADNEY: As an analog system, yes.

24 MR. KEMMERER: I don't care if it's analog or
25 digital, it's going in there.

1 MR. GLADNEY: But you want it to. That's how we
2 are doing the surveillance, we are injecting an analog value
3 to make sure the calibration of that front end is giving us
4 a true output.

5 MR. KEMMERER: Then, the two processors are not
6 isolated. I don't care where it's analog or digital, it's
7 going in and not isolating it.

8 MR. GLADNEY: Yes, but where it goes matters.

9 MR. LEWIS: You can't communicate with an isolator
10 failure.

11 MR. GLADNEY: No. That level has to be there. I
12 can't over those analog lines, send back in code to the
13 processor. But I do send in an analog signal, and that
14 analog board on the front end is electrically isolated going
15 into the processor.

16 MR. KRESS: The processor has to know that this is
17 a test signal.

18 MR. GLADNEY: No, it does not.

19 MR. CATTON: All he is doing is switching from one
20 source of analog input to the other, and the other is his
21 test. You do the same thing on the other end?

22 MR. GLADNEY: Yes, we do, on the bistables.

23 MR. CATTON: This thing is isolated. He's sending
24 in an analog signal.

25 MR. KEMMERER: The idea is you have this code in

1 there --

2 MR. CATTON: Then you activate it with this.

3 MR. KEMMERER: I go through the tester, I send in
4 the right analog signal to the right sequence of analog
5 signals, such that one sees that it doesn't shut it down.

6 MR. KRESS: It pulls out all the control rods.

7 MR. CATTON: I think it certainly could do that.

8 MR. KEMMERER: It's not isolated. You have a
9 danger there that maybe that same signal might come through
10 -- naturally and melt it when you don't want it to, but it's
11 there. I am not trying to make a big thing about this. You
12 brought it up, to my surprise.

13 The problem that occurs -- and I saw this at a DOE
14 lab also, where you have to be very careful about saying we
15 don't have to worry about it, we are isolated. You don't
16 have to worry about it, the connection is too simple.

17 MR. GLADNEY: The trojan horse scenario, clearly,
18 the only way we can solve that problem is provide guidance
19 to specify any V&V or design program considers these aspects
20 as they write the code.

21 MR. LEWIS: You really can't, because people get
22 out of the input domain you are talking about. The analog
23 signals, I don't know what happens if you drive them
24 nonlinear, if you overload the circuits. I don't know what
25 the interaction between circuits is.

1 I once saw a demo from a real spook shop, you
2 know, people who are professionals about this kind of thing,
3 about lock picking. They had a wonderful dog and pony show
4 in which they picked locks as fast as you could throw locks
5 at them. They produced one of these new, wonderful magnetic
6 locks that have no access. There are no holes in the darn
7 thing but you have a magnetic key which lines up the
8 tumblers and poof, it opens. It is therefore pick proof
9 because you can't get inside.

10 These guys said we wondered about this one for a
11 day but we found that if you take a little hammer and hit it
12 right here it pops open, and indeed it did. That's
13 extending the conversation. That's going outside the domain
14 people -- everyone was thinking of ways to imitate the
15 magnetic signal and they found that banging it open worked.
16 You have to have a spook mind to be a spook. It's just an
17 admonition.

18 Thank you, very much. We have reached a point at
19 which we are scheduled to have a conversation but we have
20 been having a conversation all day. I think we can go off
21 the record now, if that's okay.

22 [Whereupon, at 4:05 p.m., the transcribed portion
23 of the meeting concluded.]
24
25

REPORTER'S CERTIFICATE

This is to certify that the attached proceedings before
the United States Nuclear Regulatory Commission

In the Matter of:

NAME OF PROCEEDING: ACRS Computers in Nuclear Power
Plant Operations

DOCKET NUMBER:

PLACE OF PROCEEDING: Bethesda, Maryland

were held as herein appears, and that this is the
original transcript thereof for the file of the United
States Nuclear Regulatory Commission taken by me and
thereafter reduced to typewriting by me or under the
direction of the court reporting company, and that the
transcript is a true and accurate record of the
foregoing proceedings.

Michael D. Paulsen

Official Reporter
Ann Riley & Associates, Ltd.

REPORTER'S CERTIFICATE

This is to certify that the attached proceedings before the United States Nuclear Regulatory Commission

In the Matter of:

NAME OF PROCEEDING: ACRS Computers in Nuclear Power
Plant Operations

DOCKET NUMBER:

PLACE OF PROCEEDING: Bethesda, Maryland

were held as herein appears, and that this is the original transcript thereof for the file of the United States Nuclear Regulatory Commission taken by me and thereafter reduced to typewriting by me or under the direction of the court reporting company, and that the transcript is a true and accurate record of the foregoing proceedings.

Mary C. Larkin

Official Reporter
Ann Riley & Associates, Ltd.

Applying Cleanroom Engineering To The Development and Certification of Correct Software for Shutdown Systems in Nuclear Power Plants

Testimony Prepared for
February 9, 1993 ARCS Meeting
Digital Systems Safety for Nuclear Power Plants
USNRC Advisory Committee on Reactor Safeguards
Computers in Nuclear Plants Subcommittee

Prepared by
Richard H. Cobb and Harlan D. Mills
Software Engineering Technology, Inc.
Lanham, Maryland
(301) 731-6200 email cobbr@source.asset.com



SOFTWARE ISSUES TO BE DISCUSSED

1. Is it feasible to produce correct software? Correct software provides its users with failure-free behavior.
2. What engineering practices should software developers follow to increase the probability of producing correct software?
3. Is it is feasible to specify a correct function?
4. What engineering practices should software developers follow to increase the probability that the function defined in the software specification is "correct"?
5. What is the probability that software satisfying the stringent reliability requirements can be efficiently produced using the suggested practices?
6. What methods should regulators follow to determine the likelihood that the software to be licensed is correct so it will provide failure-free performance?
7. What is the probability the regulators will make a licensing error when employing the recommended techniques?

CONCLUSIONS

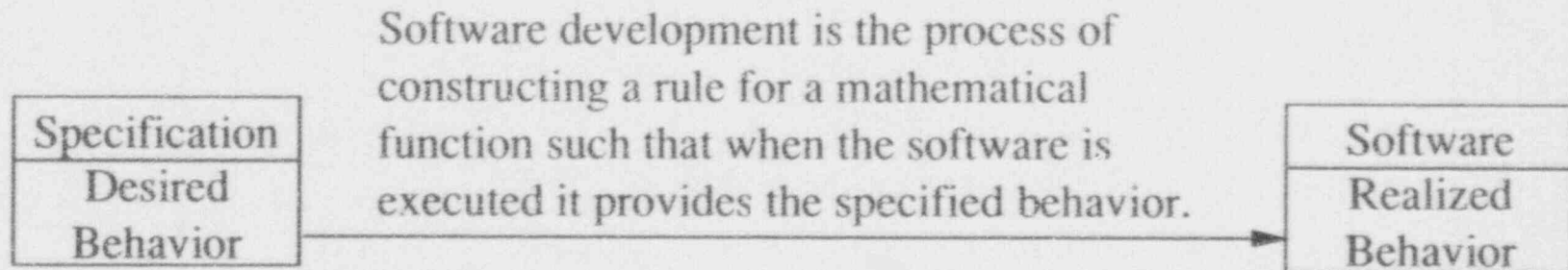
1. Correct (failure-free) software can be produced and verified.
2. Correct functions can be developed and verified for the software components.
3. A quantitative certification process exists to assess the quality of the end product.
4. The process used to develop the functions and the software can be reviewed and regulated by the NRC.
5. Complete verification proofs and certification tests can be reviewed and regulated by the NRC.
6. Software can be the most reliable part of a shutdown system.

These claims are not based on wishful thinking but are based upon the application of sound engineering practices which are rooted in a firm mathematical foundation. The principal inventor of these ideas, Dr. Harlan D. Mills has named the set of these practices Cleanroom Software Engineering.



PRODUCING CORRECT SOFTWARE IS FEASIBLE

Correct software provides exactly the behavior defined in its specification. Correct software provides failure-free performance.



The craft-based practices in common use today are not adequate for software development since the cut-and-try practices frequently result in error prone code.

Engineering-based methods for software development based on a sound science foundation are now available. Such methods can be used to develop a correct rule, expressed in the chosen programming language, for a given function.

A program is nothing more or less than a mathematical writing that can be manipulated, analyzed and reasoned about.

THE SIMPLER THE REQUIRED BEHAVIOR THE EASIER IT IS TO DEVELOP AND VERIFY THAT THE SOFTWARE IS CORRECT

Since that designers of shutdown systems only require its software components to perform single, clear functions, it will be easier to develop and verify the correctness of the shutdown software than it is for most other software projects where the software is being asked to perform a multitude of more complex functions.

In most project situations the emphasis is on developing a correct program, not on developing a proof of correctness. In this situation the stress is on developing a correct program and recording the proof of correctness so all interested parties can evaluate the proof to insure the programs provide correct behavior.

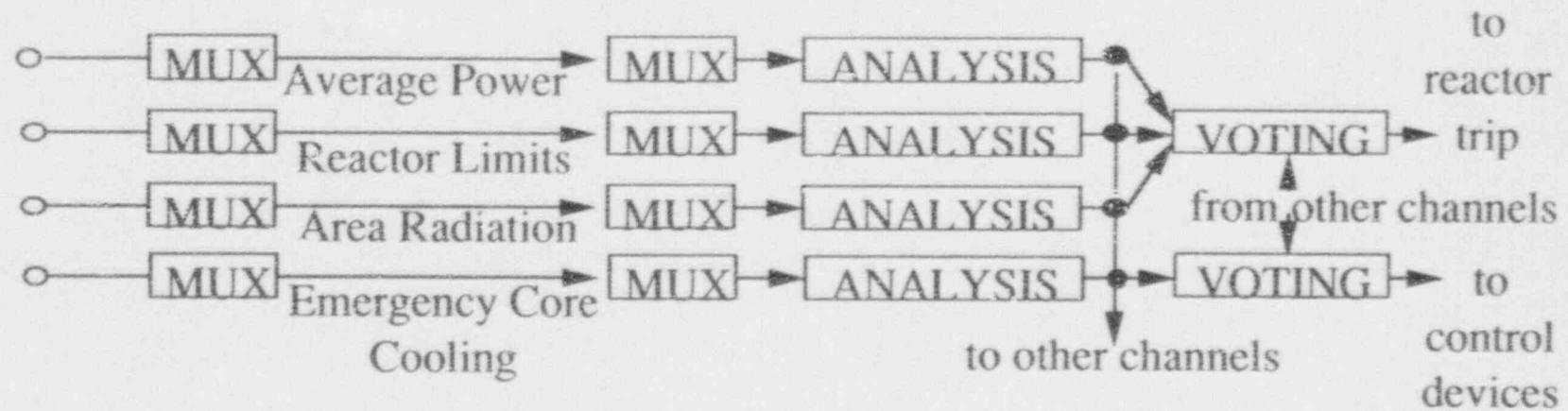
The effort of documenting a complete proof of correctness is directly proportional to size of the program. In this situation since the designers are planning for small, simple straight forward programs, the task of preparing complete proofs of correctness is a reasonable effort.



A SIMPLIFIED VIEW OF SITUATION

Typical shutdown channel (1 out of 4)

(Not in full detail and not representative of any single manufacturer.)



Key points:

Components per channel - 14+ (more like 20 due to multiple data acquisition lines)

Total code per channel - 12 - 20 KLOC

Each software item provides a small, single, well defined function.

No requirement for operating system, DBMS or other error prone COTS software

There are examples of such small well defined software systems that have been carefully developed and formally proved that are providing failure-free performance over years of heavy usage.



ENGINEERING PRACTICE FACILITATE THE DEVELOPMENT OF CORRECT SOFTWARE

It is our opinion that in 1993 the software industry is capable of engineering failure-free software for the shutdown application and the NRC can verify that the software industry has in fact engineered the required software. Furthermore, we believe the software can be the most reliable element in the shutdown channel. These beliefs are not based on wishful thinking but on the application of rigorous engineering practices which are rooted in sound mathematical theory.

We call these engineering practices Cleanroom Software Engineering. Software is designed using sound mathematical principles in a series of small refinements each followed by a verification. The result is that design faults are kept out so they do not have to be found by testing and then removed. Thus we make the analogy to cleanrooms which electrical engineers use to keep dirt and other contaminants out of precise assemblies.

In the next slides some of the differences between craft-based development and engineering-based development are highlighted.



Craft-based Development

A program is regarded as lines of instructions.

Programs are normally recorded using structured constructs.

A specification is regarded as a list of requirements that leave many details for latter invention. Specifications are often considered to be a burden.

No formal transformation process is known or used. Debugging is used to verify code.

Failures are expected and accepted.

Programs are regarded as private art.

Engineering-based Development

A program is regarded as a correct rule for a function.

Programs are always recorded using structured constructs.

A specification is regarded as a clear, precise description of desired behavior. Black box functions define behavior. No details are omitted. Specifications are expanded to code when they are complete.

The specified function is transformed into code in a series of small refinements. Each refinement is followed by a proof.

Failures are considered to be unacceptable.

Programs are regarded as public literature.



Craft-based Development

Testing is used as a means to find program faults.

Tests are designed to cover all program paths. This is an impossible goal. Anecdotal arguments are used to extend observed results to the population.

Projects are organized around individual craftspersons performing activities.

Projects are often not under the intellectual control of the project team.

Engineering-based Development

Testing is used to confirm the hypothesis of correctness.

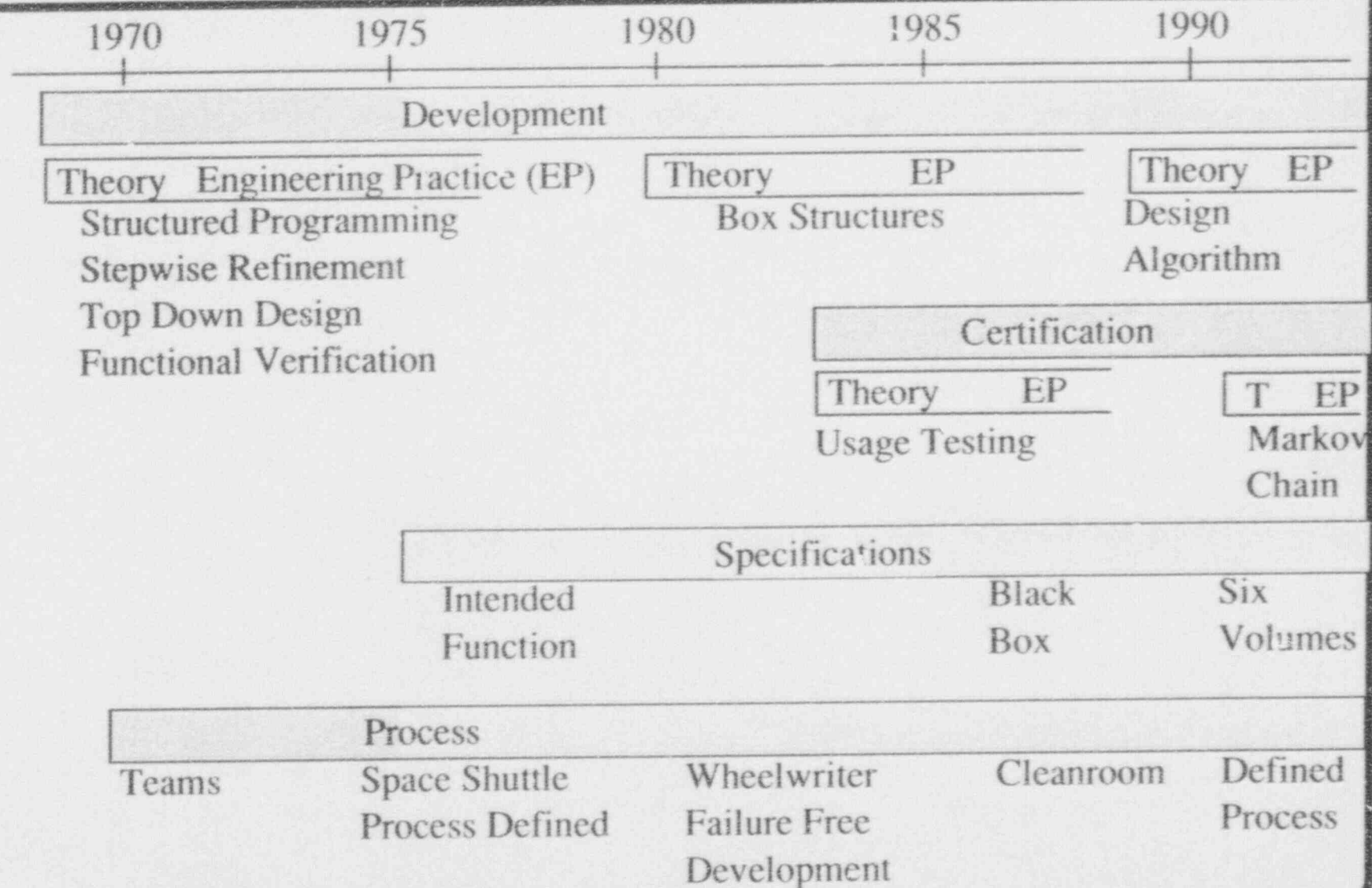
Software is evaluated using a random sample of expected usage. A Markov model is used to define the probability density function. Sample reliabilities are used to estimate population reliabilities.

Projects are organized around engineering teams producing documents.

Projects are organized to keep them within the intellectual control of the project team.



THE CLEANROOM PRACTICES HAVE BEEN DEVELOPED AND REFINED OVER 25 PLUS YEARS



REDUCING THEORY TO PRACTICE HAS RESULTED IN MEANINGFUL RESULTS

Development Practice	Development Defects/KLOC	Operational Defects/KLOC	Productivity LOC/Staff Mo
Traditional (unstructured programs, bottom-up design)	50 - 60	15 - 18	unknown or unmeasurable
Modern (structured programs, top-down design, code inspections)	20 - 40	2 - 4	75 - 475
Early Cleanroom (structured programs, top-down design, functional verifications, usage testing)	< 5	< 1	700 - 750
Later Cleanroom (early Cleanroom plus box structures, Markov chains, improved specs & process)	< 5	<< 1	> 750



SOFTWARE AND SPECIFICATION FAILURES (1 of 2)

A failure occurs when the system does not provide the required service.

A specification failure occurs when a system does not provide the required service because the system designer did not account for some circumstance or otherwise specified an improper or incomplete function to describe the desired behavior.

A software failure occurs when the system does not provide the required service because the software designer introduced an unintended deviation from the specified behavior.

Software systems failures also occur when the software designer and the system designer do not fully communicate because the specification is imprecise, incomplete or otherwise misleading.



SOFTWARE AND SPECIFICATION FAILURES (2 of 2)

Since most software projects lack a rigorous process it is very difficult to classify failures in to one of following categories:

Software failures are due to initial design

Software failures are due to changes to a previous design

Specification/Software failures are due to imprecise specifications

Specification failures are due to wrong or incomplete specifications

We do know that most software failures can be traced to changes to a previous version of the software. The cause of the change may in response to a correction to a previous design error, a reinterpretation of a system requirement or in response to a specification shortcoming. Careful observations indicate that between 1 time in 10 and 1 time in 20, say 15% of the time, that when software is changed the result is the introduction of a fault that can result in a failure. These observations are made in organizations that are regarded as examples of the best software development organizations.



SOFTWARE AND SPECIFICATION FAILURES: A RECOMMENDATION

In evaluating issues relative to potential system/software safety failures, carefully identify each potential source of failure and take appropriate action to mitigate the possibility of introducing failures from each source. For example, clearly separate software development issues and specification preparation issues.

Specifications define desired behavior to provide some service by defining a function that describes the desired behavior.

Software is a rule that will provide the desired behavior when it is installed on the specified processor by accepting a stimuli (argument) and issuing a correct response (value) according the intended function contained in the specification.



CONCLUSIONS/RECOMMENDATIONS FOR FIRST TWO ISSUES

1. Is it feasible to produce correct software?
2. What engineering practices should software developers follow to increase the probability of producing correct software?

It is feasible to produce correct software.

The probability of producing correct software is greatest if a rigorous mathematical based engineering method like Cleanroom Software Engineering is used. If correct software is desired, minimize changes to the software.

DEVELOPING SHUTDOWN SOFTWARE

For each software item

Develop software from the black box function contained in the approved, complete, rigorous specification.

Develop software top-down, in a series of small refinements.

Verify each refinement.

Record each refinement in the design hierarchy.

The complete design trail must present a full and complete proof argument that the software will provide exactly the behavior defined in the specification.



CERTIFYING SHUTDOWN SOFTWARE

For each software item certification will begin when the development team is satisfied the software is correct.

Certify the software using a statistical significant sample. Generate test scenarios using a Markov model which can be used to constructively generate the probability density function of use.

The certification team will establish the software for execution.

The certification team will then run sufficient failure-free executions to enable the software to be certified to the desired reliability.

If any failures are encountered, the software should be rejected if it is found that fault causing the failure is not the result of simple mathematical fallibility.

Any failure due to mathematical fallibility should be fixed by the development team and all verifications must be recertified.



CERTIFYING THE SHUTDOWN SYSTEM

When all the individual components have been individually certified, assemble the system and

Certify the software using a statistical significant sample. Generate test scenarios using a Markov model which can be used to constructively generate the probability density function of use.

The certification team will then run sufficient failure-free executions to enable the system to be certified to the desired reliability.

If any failures are encountered the system should be rejected if it is found that fault causing the failure is not the result of simple mathematical fallibility.

Any failure due to mathematical fallibility should be fixed by the development team and all verifications recertified. The component should be recertified.



SPECIFICATION CHALLENGES

Specifying the desired behavior for the shutdown channel.

Insuring the specified channel behavior considers all safety related possibilities.

Allocating a mission to each of the channel components.

Preparing specifications which define each component's behavior.

Verifying that the composite behavior of the behavior specified for each of the components is equivalent to the specified behavior for the shutdown channel.

Selecting or building components to minimize the possibility that a common mode failure overrides the redundancy built into the channel in such a way that the system fails to function when it should.

These challenges confront the shutdown system designer no matter if software components are included in or excluded from the design. The presence of correct software only enhances the design choices and makes it possible for the designer to specify a better solution.



SPECIFYING CORRECT BEHAVIOR FOR A SHUTDOWN SYSTEM

The problem is the same if software components are included or not included.

All safety issues must be clearly identified and mitigated by the specified design.

The specification task is considered more manageable by knowledgeable authorities on reactor operations if software is included due to the ability to specify improved control logic and its dependability since it cannot wear out prematurely.

Use black box functions to specify desired behavior for the system and for each component. *(See next slide for information on black box functions.)*

Record the specification for the system and for each component of the system in an appropriately modified version of the six volume Cleanroom specification. *(See second next slide for high level outline of six volumes.)*



BLACK BOXES: SOME FUNDAMENTALS



A **black box** is a mechanism that accepts stimuli and for each stimulus, produces a response before accepting another stimulus; furthermore, each response is uniquely determined by the history of stimuli accepted by the black box.

Every system exhibits black box behavior.

Black box transition

Stimulus History \longrightarrow Response

External behavior only

The user view

No state (data), no procedure

A CLEANROOM SPECIFICATION

Volume	Purpose
Mission	A precise statement of the requirements
User's Reference Manual	A definition of the stimuli and responses invented so the system can fulfill its mission
System Function	A black box function that defines system responses in terms of stimuli histories
Specification Validation	A rigorous argument that the system as defined will meet its defined mission
System Usage Profile	The probability density function for system usage in terms of a Markov model stating the probability of moving from system state i to system state j
Construction Plan	A plan for building the system in a series of increments such that each accumulation of increments is executable by user stimuli



COMMON MODE FAILURES

An important safety issue facing the designers of a shutdown system is to devise a system that is immune to common mode failures.

The concern is that a hardware design error, a software design error or a software programming error may result in a common mode or common cause failure of redundant equipment. (NRR Slide)

We would prefer the following restatement because we believe it more clearly identifies the cause of the problem and therefore a search for solutions:

The concern is that a hardware specification failure, a hardware manufacturing failure, a software specification failure or a software implementation failure may result in a common mode or common cause failure of redundant equipment.



COMMON MODE FAILURE DEFENSES

The defenses against common mode failure are

Quality and
Diversity

Defenses against software implementation failures:

Quality Use rigorous methods to develop software
 Develop and record complete proofs for the software
 Certify the software

Diversity Develop, verify and certify a separate rule for some (all) components

Defense against software and hardware specification failures:

Quality Use black box functions
 Develop a complete specification
 Review specifications with a jury of peers

Diversity Not analyzed

Defenses against hardware implementation failures:

Not analyzed



CONCLUSIONS/RECOMMENDATIONS FOR SECOND TWO ISSUES

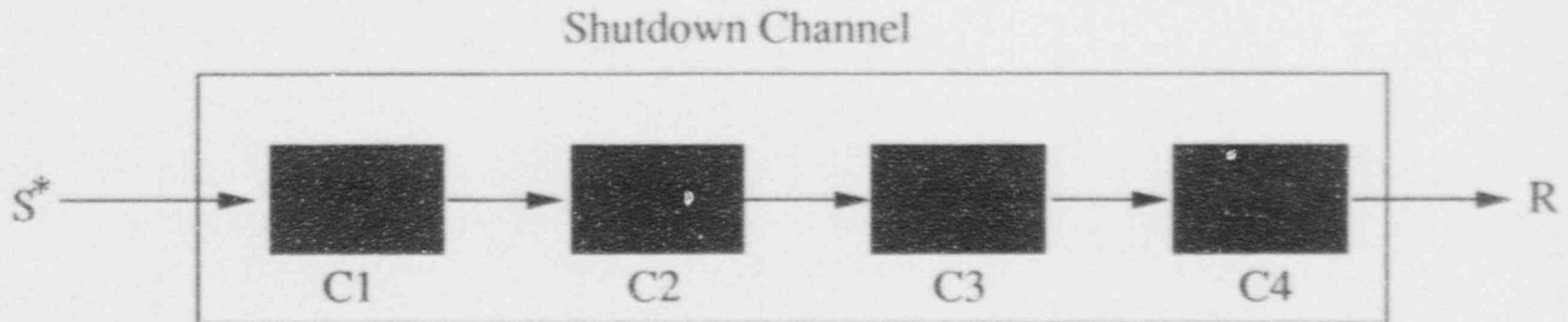
3. Is it is feasible to specify a correct function?
4. What engineering practices should software developers follow to increase the probability that the function defined in the software specification is "correct"?

If it is possible to define when a power plant is to be shut down because it is entering an area of unsafe operation then it must be feasible to specify a system that will perform the desired mission including minimizing the probability of common mode failures.

The designers should make use of black box functions to record the desired function for the system and each of its components. The specifications should include all features of the Cleanroom six volume specification.

The NRC should license the design prior to its implementation. The implementation can then be licensed when it is demonstrated the implementation provides the specified behavior.

VERIFYING COMPONENT SPECIFICATIONS



The black box behavior for the shutdown channel can be derived by combining the black box behavior for C1, C2, C3 and C4. The derived black box behavior can then be compared to the intended black box behavior for the shutdown channel. In this way the allocation of function to the individual channel components can be verified.

FEASIBILITY OF ECONOMICAL DEVELOPMENT OF THE REQUIRED SHUTDOWN SOFTWARE

Most of the required effort is necessary even if the system does not use software.

The inclusion of software components causes concern for two reasons:

Hardware engineers have had a history of success so we do not worry.

Software developers have a history of confusing specifications preparation with software implementation. They also have a history of using inadequate practices to develop software. In summary, software developers have a poor track record.

Using the practices associated with Cleanroom Software Engineering software developers can efficiently produce high quality software.



RECOMMENDED PROCEDURE FOR DEVELOPING AND CERTIFYING A SHUTDOWN SYSTEM

Fully and completely specify the desired behavior for the system and the desired behavior for each component prior to initiating software development. Desired behavior should be specified using a black box function to facilitate the full and complete communication of the intended behavior.

If software is a component of the system, develop the software top-down by refining the specified function into a rule that will provide the specified behavior when it executes on the specified computer. Verify each refinement using functional verification. Independently certify behavior by using statistical usage testing. Reject the software implementation if too many failures are found or any failure that is not the result of mathematical fallibility is encountered.

Integrate and certify the system behavior by using statistical usage testing. Reject the system if too many failures are found or any failure that is not the result of mathematical fallibility is encountered.

The NRC should license the design and the implementation of the design separately.



PROPOSED DESIGN AND APPRAISAL CYCLE

Project Invocation

- Project Mission
- System Development Plan
- Software Development Plan
- Quality Plan

Process Review and Acceptance

System Design

- Specification for Shutdown System
- Specifications for Each Channel (4)
- Specification for Each Component
- Process Audit Report
- Analysis of Quality Metrics

Design Approval and Licensing

Component Development

- For each component
- Design & verification hierarchy
 - Component Test Bed
 - Certification Report
 - Process Audit Report
 - Analysis of Quality Metrics

Component Certification and Licensing

Channel Integration

- Channel Test Bed
- Certification Report
- Process Audit Report
- Analysis of Quality Metrics

Channel Certification and Licensing

OUTLINE OF SPECIFICATION DOCUMENTS

Specification for Shutdown System

- Mission
- System Function
- System Components
- Allocation to channels
- Safety Analysis
- Validation Argument
- System Usage Profile
- Construction Plan

Specifications for Each Channel (4)

- Mission
- Channel Function
- Channel Components
- Allocation to components
- Safety Analysis
- Validation Argument
- Channel Usage Profile
- Construction Plan

Specification for Each Component

- Mission
- Software Function
- Hardware
- Stimuli/Responses
- Validation Argument
- Usage Profile
- Construction Plan



METHODS THE NRC CAN FOLLOW TO DETERMINE IF A LICENSE SHOULD BE GRANTED

For Process Review

The plans should clearly separate specification preparation and software development.

The development method should be rigorously based on the mathematical foundations of software.

The development method should minimize debugging.

The process should be clearly identified.

For Design Approval and Licensing

The specifications should be complete and include black box functions and usage profiles. Otherwise, the NRC should use the same means they use for other systems.

For Component Certification and Licensing

Evaluate process and failure history.

Evaluate a sample of proofs. If any proof is found to be in error reject software. (A 100% sample is possible.)

Evaluate certification history.

For Channel Certification and Licensing

Evaluate process and failure history.

Evaluate certification history.



ADEQUACY OF PROPOSED PROCEDURES

Relative to implementation correctness:

The proposed procedures will identify systems/software that are failure prone.

The procedures used should be geared to reject marginally acceptable systems. To do this it will be necessary to prepare a detailed certification plan.

If the recommended software development methods are followed, the software should be the most reliable element in the shutdown channel.

Relative to specification correctness:

All human activity is subject to fallibility and therefore, design omissions are possible. That is why effort must be focused on design specifications and the design should be the subject of a separate licensing action.



WHAT IS THE PROBABILITY THE REGULATORS WILL MAKE A LICENSING ERROR ?

	Relative to Specifications	Relative to Implementation
Type 1 Error: Accept a system that should be rejected	What is the likelihood of a design oversight? *	Very Low
Type 2 Error: Reject a system that should be accepted	Very Low	Low

* We believe this is the vital issue that Professor Leveson has been attempting to get software industry to focus on but we think her message sometimes gets lost because of a lack of a clear distinction between specification failures and software failures.



TOPIC 1: IS THE SIZE/COMPLEXITY OF NUCLEAR SAFETY-RELATED SOFTWARE SUFFICIENTLY LIMITED TO PERMIT EFFECTIVE USE OF FORMAL V&V METHODS?

Yes. The proposed designs decompose software items into small blocks each with a well defined, self-contained function. The software development, verification and certification issues all well in the bounds of what we can expect to accomplish with Cleanroom methods as they are understood and applied in 1993.

The current craft-based practices that are in wide spread use today are not acceptable for most applications to which they are being applied and certainly not for the application being discussed at this hearing.



TOPIC 2: OVERALL ASSESSMENT OF THE BEST METHODS FOR SOFTWARE CERTIFICATION/V&V IN TERMS OF EFFECTIVENESS, EFFICIENCY, COMPREHENSIVENESS AND DEGREE OF ASSURING HIGH RELIABILITY/FIDELITY IN SOFTWARE SYSTEM PERFORMANCE

If humans were infallible, the proofs prepared during development should guarantee the software would provide correct performance.

Since humans are fallible subjecting the proofs to independent peer review will increase the likelihood of detecting a fault. The use of statistical testing will also provide an added measure of assurance.

Using the recommended methods the question will not be:

Does the software behave as specified?

The question will be:

Is the specified behavior adequate to perform the defined mission?



TOPIC 3: ENDORSEMENT OF SPECIFIC SOFTWARE DESIGN CRITERIA AS PROMULGATED BY PROFESSIONAL SOCIETY STANDARDS (E.G., ASME NQA2, IEEE/ANSI-ANS 7-4.3.2)

In general, software development standards codify craft-based practices and therefore are insufficient for developing quality software.

The methods and practices we are recommending are sufficient for the implementation of software systems to satisfy the specified mission.



TOPIC 5: APPROACHES TO MANAGING SOFTWARE RISK BOTH IN SOFTWARE DEVELOPMENT AND IN SAFETY SYSTEM OPERATION

Use the highest quality software development practices possible. Avoid all high risk activities.

Carefully prepare and evaluate the specifications from a safety analysis standpoint.

Designers want software functionality to specify a more robust system since software provides functionality that hardware devices cannot provide.

Designers want software devices because

software components do not fail to operate when required as hardware components do.

systems which contain software components can provide self assessment so the premature failure of hardware devices can be detected as soon as they occur.

In general, software provides system benefits. Craft-based development methods introduce implementation risk. Engineering-based development methods mitigate the software implementation risk.



DEVELOPING THE SHUTDOWN SYSTEM REQUIRES

1. Specifying the desired behavior for the shutdown system.
2. Insuring the specified behavior is complete.
3. Defining the components that comprise the shutdown channel.
4. Allocating a mission to each of the channel components.
5. Specifying the behavior for each component.
6. Verifying that the composite behavior for all the components provide the same behavior as the specified behavior for the entire system.
7. Developing each component by:
 - Implementing the specified behavior.
 - Documenting complete proofs of correctness.
 - Certifying each component provides failure-free behavior.
8. Assembling the channel.
9. Certifying the channel provides failure-free performance.

Cleanroom methods provide the means to accomplish steps 6, 7 and 9 with engineering rigor. Cleanroom methods also provide the means to support the steps 1, 2, 4 and 5 due to the rigor provided by using black box functions in specifications.



CONCLUSION

Software components provide substantial benefits to the designer of control systems.

The risk associated with specifying software components is that the delivered components may include implementation faults that can result in a failure prone operation. Due to the youth of the software industry such flawed software is common.

The software implementation risk will be mitigated when software solutions are engineered instead of crafted.

The engineering methods we have proposed permit the industry to develop verified, certified failure free software components.

In this way shutdown channel designers can obtain the benefits offered by software components.



ACRS Meeting
Digital Systems Safety for Nuclear Power Plants

USNRC Advisory Committee on Reactor Safeguards
Computers in Nuclear Power Plants Subcommittee
February 9, 1993
Phillips Building Room 110
7920 Norfolk Avenue
Bethesda, MD 20814

Applying Cleanroom Engineering To
The Development and Certification
of Correct Software for
Shutdown Systems in Nuclear Power Plants

Software Engineering Technology, Inc.
Richard H. Cobb, Harlan D. Mills and James A. Whittaker



SOFTWARE ENGINEERING TECHNOLOGY, INC.

4600 Forbes Blvd., Lanham, MD 20706 • Phone (301) 731-6200 • FAX (301) 731-6203
1918 Hidden Point Road, Annapolis, MD 21401 • Phone (410) 757-7149 • FAX (410) 757-4389
Corporate Office: 2770 Indian River Blvd., Vero Beach, FL 32960 • Phone (407) 569-3722

The purpose of this paper is to amplify the testimony presented by Software Engineering Technology, Inc. (SET) at the February 9, 1993 meeting of the USNRC Advisory Committee on Reactor Safeguards and the Computers in Nuclear Power Plants Subcommittee by providing additional technical details not covered in the 80 minutes provided.

1.0 Outline of Testimony

In the executive summary, seven facets of the issues that are relevant to the subject of the hearings are identified. Included also are a summary of recommendations. Following the recommendations, a brief history of software development is presented to put the recommendations into perspective. During this discussion aspects of the software development effort for a shutdown system are discussed. Several appendices provide additional information that are relevant to the deliberations of this subcommittee.

2.0 Executive Summary

The objective of this testimony is to provide information on mathematical results and associated engineering practices that permit the specification, development, and certification of highly reliable software. Our testimony recommends that manufacturers specify, develop, and certify shutdown software using Cleanroom Software Engineering practices. The Cleanroom engineering and management processes and practices will enable manufacturers to demonstrate, to the full satisfaction of the NRC, that the software they provide meets the required reliability levels.

The nuclear industry desires to employ software components in advanced digital I&C systems in order to exploit the significant control benefits provided by software technology. These benefits include (1) the ability to replace traditional set point logic, implemented in arrays of relays, by algorithmic approaches which offer better results, (2) continuous on-line testing, (3) self-diagnostic aids, (4) high accuracy and drift free operation, and (5) use of signal multiplexing and fiber optics for isolation. Only the use of software technology will provide these benefits. Unfortunately, the software industry does not have a sterling record when it comes to developing failure-free software. The NRC is, justifiably, concerned that the price of gaining the benefits may not be worth the risk. However, as this testimony will show, the software industry is capable of engineering failure-free software for the shutdown application and, further, that the NRC can verify that this has been accomplished. Moreover, the software components can be the *most reliable elements* in the shutdown channel. This testimony shows that these beliefs are not based on wishful thinking but on the application of rigorous engineering practices, rooted in sound scientific and mathematical theory.

2.1 Software Issues Relative to the Shutdown System

The NRC must ascertain the reliability of shutdown software to ensure that it satisfies the stringent reliability requirements placed on systems deployed in nuclear power plants. The facets of the issues relating to this task that we will discuss are:

1. The feasibility of producing correct software. **Correct Software** is software that provides exactly the behavior defined in its specification. Correct Software provides **failure-free** performance.
2. The feasibility of specifying a correct function. A **correct function** is a function which defines the behavior of a software system such that if software exists which exactly provides the defined behavior then that software will fulfill its mission to the full satisfaction of all stakeholders in the software.
3. The identification of engineering practices that software developers must follow to increase the probability that they will produce correct software.
4. The identification of engineering practices that software developers must follow to increase the probability that they will prepare a correct function for inclusion in the software's specification.
5. The probability that software satisfying the stringent reliability requirements, required by society for nuclear power plants, can be efficiently produced using the suggested engineering practices. **Software reliability**, is defined to be the probability that a given execution of a software system results in failure-free behavior. The complement of the reliability is the probability that an execution produces a failure. For example, a reliability of .99999 indicates the probability of failure is 10^{-5} or that the software is expected to fail less than 1 time in 100,000 uses. A **use** for the shutdown system is a combination of stimuli that result in the requirement that the plant be shutdown.
6. The identification of methods that regulators should follow to determine the likelihood that the software they are being asked to license is correct software and, therefore, will provide failure-free performance.
7. The probability that regulators will make a licensing error when employing the recommended techniques. There are two types of possible licensing errors (i) a "type 1 error" is made when the regulators accept (license) software that they should have rejected (because, in reality, the software is not sufficiently reliable) and (ii) a "type 2 error" is made when the regulators reject software that they should have licensed (because, in fact, the software was sufficiently reliable).

Before summarizing our conclusions relative to these seven issues, the nature of the software to be developed for a shutdown system must be considered as it impacts each decision that the NRC faces. This is the topic of the next section.

2.2 An Overview of the Shutdown Software: It is a Small, Self-Contained Application

The design of the shutdown software is significantly less complex than many other software

systems being developed today. The design considerations involved make it much easier to write correct software and to prove correctness for the shutdown system. The factors that contribute to this simplicity include the following.

- Because of its size and isolation, the entire shutdown software can be under the intellectual control of the development team. There is no need to include an operating system, a data base system, or any other form of COTS software to contribute to failure possibility. This is not typical of most software systems.
- The software system is small, less than 20,000 lines of code spread over more than 14 processors in each shutdown channel. There are many examples of critical components of the same size that have been developed with great care and formally proved that provide failure-free performance over years of heavy use.
- The specification of the mission of each software item is well-defined, thus, it is possible to create an accurate, clear, and rigorous specification of the function to be performed. Again, this situation is much simpler than in typical software systems. Included in the specification is a full hazard analysis and the associated measurements. This analysis is performed at the specification level so that proving the implementation resolves such issues can be achieved by developers.
- Since the entire software system can be maintained under the intellectual control of the development team and no tight time constraints exist, the common sources of failure prone software development practices can be avoided. For example, there will be no need to use arrays or dynamic memory allocation which plague many craft-based practices.
- Using the Cleanroom engineering practices, discussed in this document, it will be possible to formally prove every software construct. The effort will be less than it takes to write a typical math book.
- Using the Cleanroom engineering practices, it will be possible to develop a formal and concise model of expected usage that will facilitate an accelerated testing regime enabling a quality estimate to be achieved that exceeds the anticipated useful life of a power plant.

The major challenges facing the designers of the shutdown channel are:

- to develop a design that permits the combination of hardware and software to always work in the face of all possible conditions the shutdown channel may be affected by, and
- to develop the software and select other components such that common mode failures will not influence the effectiveness of the system.

The two defenses against common mode failures are: quality and diversity. We hope to

demonstrate that the software can be developed in such a way that it can be expected to provide failure free performance. The major question is to show how the software components are diverse. The only solution that we have been able to develop to insure diversity is

- Since there are an infinite number of correct rules for any function, a different rule can be developed (from the same specification) for each of the four shutdown channels. Although this increases the development work, the certification work is not affected. Engineering judgment will have to be used to determine if this added degree of confidence is justified.

The solution to the major two issues is to define the shutdown channel and to assign a clear precise mission to the software components of the channel and then to develop the software in such a way that one has a very high confidence that the software fulfills its assigned mission.

2.3 Summary of Conclusions

The following is a summary of conclusions relative to the seven facets facing the NRC.

1. It is feasible to develop correct software for the shutdown application.
2. It is feasible to correctly specify the function that the software must fulfill so that it can satisfy its portion of the shutdown mission as defined in the system specification. The issue is it possible to specify a function for the shutdown system in total so that it contains no omission or otherwise contains an error. This latter point is not a software issue but a system specification issue.
- 3,4. The software developers should utilize the set of software engineering practices that have been identified as Cleanroom Software Engineering (*) by its principal inventor, Dr. Harlan D. Mills, to specify, develop, and certify the shutdown software. Using these methods, the manufacturer can
 - (i) prepare a specification that fully and completely defines the mission to be assigned to the shutdown software,
 - (ii) develop essentially correct (failure-free) software (< 1 failure per 10^9 uses),
 - (iii) develop a deductive proof of software correctness,
 - (iv) perform a statistically valid experiment that permits the organization to make the claim that the control channel operates correctly at least 99,999 uses out of 100,000 uses with a confidence of 99.9%
5. The manufacturer will be able to efficiently specify, develop, and certify the software

using the recommended methods. In fact, successful application of Cleanroom will make the software among the most reliable components in the entire channel.

6. In processing licensing applications for the software, the NRC should follow the following procedure.
 - (i) Assess the software specification to ensure that the function which has been defined will, indeed, shut down the plant whenever the measurements of the physical parameters substituted into the specified equations indicate to do so. If any error is found in the specification, the entire application should be rejected. The specification assessment must include an evaluation of the engineering solution to the common mode failure problem. At the same time, all identified safety hazards, and the planned solutions, must be assessed.
 - (ii) Assess the proof of correctness for the software. This assessment can be carried out in a number of ways. For example, the commission's staff can take a random sample (say 10%) of the individual functional verifications that make up the entire proof and have them presented to a jury of peers or have them performed by an independent organization. If no mistake is found, the proof could be accepted. The commission could, of course, take any size sample they feel necessary, including a 100% sample. If any mistake is found in any of the proofs, the entire application should be rejected. When the software development organization utilizes the Cleanroom practices the complete design trail is always available.
 - (iii) Assess the test suite that was executed to ensure they were performed correctly and do, indeed, verify that the channel provides the claimed reliability.
 - (iv) Combine all evidence to accept or reject the licensing application.
7. We believe that if the NRC utilizes the recommended methods the staff will be able to reach an informed conclusion that minimizes the probability of either accepting software they should reject or rejecting software they should accept.
- * Appendix A provides a brief overview of the Cleanroom engineering practices while Section 3 provides a brief history of software development practices to put into perspective the difference between Cleanroom practices and the craft-based software development practices you have heard about in the previous sessions of these hearings. Our hope is that in this way you will be able to understand why Cleanroom practices provide the necessary capabilities to meet the stringent reliability requirements of shutdown software.

In some cases, a vendor may already possess software that they believe to be correct. Similar mathematical procedures can be used when determining to license such software. If the vendor

does not have a specification in the required format, it will need to be created. They most likely will need to prepare the required correctness proof using function abstraction techniques. Testing may have to be repeated. In any case, the recommended procedure to support licensing will not be burdensome if the software is, in fact, correct. It will be burdensome if the software is not correct because the recommended process will find the flaws in the software; which, of course, is the purpose of the process.

2.4 Application of Recommended Methods In Other Industries

One obvious question is that if these ideas are so good, why isn't everybody using them and why haven't earlier testimonies identified them?

As will be seen in the next section, some of the engineering practices that comprise Cleanroom, have been in use since the 1970s. The structured programming concepts have improved average quality across projects from small to large by a factor of 5. For those installations that included the functional verification element, the improvement was a full order of magnitude. Research is continuing to perfect the mathematical basis and associated engineering practices for Cleanroom. The improvement since 1990 has been significant. The ideas that complete the entire Cleanroom process, as recommended for use for developing shutdown software, were first specified in the 1989/1990 time period and have been perfected in the last several years. Cleanroom is a complete software engineering discipline that is comprised of field-proven techniques (many of which have been borrowed by other technologies) and newly developed methods.

SET started to actively promote the wider use of the Cleanroom practices in 1991. Organizations that require reliable software were targeted for obvious reasons. We have a series of successes with DoD agencies and telecommunications companies and are continuing to work with organizations in both areas. With the introduction of the new energy bill, the rebirth of the nuclear industry may be at hand. Therefore, the NRC staff was contacted. They showed genuine interest and recommended that we present our ideas to the manufacturers who will be developing the software. They provided us with names of each potential manufacturer. We have talked to all of them and have received a positive response about the potential that the Cleanroom technologies offer. At the suggestion of the manufacturers the ARCS committee was contacted. That is why we are here today.

It is our belief that the Cleanroom ideas can help the nuclear industry develop the right solution for this critical application.

3.0 Software Development: A Short History

Society has been developing software for a relatively short period of time, say 40 years, or about one human generation. In this short time, a vigorous commercial industry has emerged. The industry is based on programmers working as craftspersons. As more and more critical applications come under software control, the need to move away from craft-based software

development to engineering-based practices is becoming paramount. It is the destiny of software development to become a true engineering discipline with theory and practice similar to that of other engineering disciplines. In this section, we will review the progress being made toward developing an engineering-based practice for software development.

The basic problem, with which the NRC and the nuclear industry is struggling, can be stated as follows:

Is it possible to develop software for inclusion in safety-critical applications (e.g., nuclear power plants) using craft-based approaches to software specification, development, and certification?

The answer must be an emphatic NO! Software destined for any application, especially a critical application should not be crafted, it should, and must, be engineered. Civil engineers do not build bridges by applying a craft, they do it through the application of sound engineering practice that is supported by scientific and mathematical theory. The same should be true of constructing software, which can have as many or more lives depending on it.

Society has found, through experience, that it is quite difficult to construct failure-free software using craft-based practices. The difficulty and frustration that users experience when software fails is an all too familiar experience.

During this first human generation of software experience, there have been those who have been developing scientific foundations to support the emergence of a software engineering practice for software specification and development. These individuals and their associates have been expanding the current state of practice using these ideas.

3.1 Structured Programming

During the past 40 years, society has made enormous strides in improving the general level of craft-based approaches as a result of integrating some of the scientific ideas. The idea that has had the most significant impact in improving the ability to develop correct software is most certainly structured programming and the associated design practice of recording a structured program in a top-down, systematic fashion.

A structured program is a program that is written using only sequence (i.e., `do ... od;`), alternation (i.e., `if ... then ... else ... fi;`), iteration (i.e., `do ... until ... od;` or `while ... do ... od;`), and concurrent (i.e., `con ... noc;`) control structures.

When Dijkstra in 1969 [1] suggested that a program could be written using only structured statements the idea was ridiculed -- most people thought it was impossible to write a program without a Go To statement to define the flow of logic. Leading thinkers such as Mills, Wirth, and Hoare [cf. references 4, 6, and 3 respectively] expanded the ideas and refined the structured

programming proposal to include top-down design, program verification, and correctness conditions. By the end of the 1970s, all serious computer programmers knew that it was possible and necessary to write structured programs. Programming languages were specifically designed (or redesigned) to make it possible for programmers to write structured programs. Top-down design and structured inspections became the expectation during the 1980s. Appendix A provides more information on structured programming and top-down design using stepwise refinement.

We believe data collected by Dyer and Kouchakdjian [2] shows the impact of the advent of the structured programming revolution. They found the following by inspecting metrics from a large number of programming efforts.

Development Practices	Development Defects/KLOC	Operational Defects/KLOC	Productivity LOC/Staff Mo
Traditional (unstructured programs, bottom-up design)	50 - 60	15 - 18	unknown or unmeasurable
Modern (structured programs, top-down design, code inspections)	20 - 40	2 - 4	75 - 475

Other practices followed by some software developers during the 1980s may have contributed modestly to the decrease in observed operational defects by a factor of 5+, but in our opinion the single, largest contributor to this improvement was the structured programming revolution.

3.2 Program Verification

There was one fundamental aspect of structured programming that did not catch on widely in practice. The idea was formal verification of programs. The reason the structured programming pioneers pushed structured programming was because they saw its constructs as a prerequisite to formal verification of programs. The structured programming pioneers (Dijkstra, Mills, Hoare, Wirth) were all mathematicians. Their work was organized by the realization that a program is a mathematical object because it is a rule for a function. Thus, mathematical reasoning, in the form of correctness proofs, could be used as with any other mathematical object. The ideal goal was to develop the mathematical machinery for developing a deductive proof of correctness for a program. They all saw the establishment of structured program constructs as a prerequisite for the formal verification of programs.

In creating the machinery for proving programs correct, the pioneers took two different approaches. Dijkstra and Hoare took the approach of axiomatic verification and Mills took the approach of functional verification.

Axiomatic verification proves programs correct by reasoning about the effect of programs on data. The proof takes the form of predicates on the data in various places in the program that are invariant during execution. The relationship between these predicates are given by axioms of the programming language (hence the name), and entry/exit predicates together define the program function in an alternative form.

Functional verification proves programs correct through correctness conditions based on the fact that programs describe a function rule. For each structured construct (sequence, alternation, etc.) in a program there are a small number (the most is three) of conditions that must be proved through simple set-theoretic manipulations. Replacement of programs parts by their respective function abstractions provides for proofs of very large programs in a series of small, controllable steps.

The practical difference between functional verification and axiomatic verification is that functional verification scales-up to even the largest programs. This is because no matter how large a program is, it is still a function and the mathematics apply. Large programs can be easily broken down into smaller programs (in fact, they are written that way in top-down design) which are also functions to which the mathematics apply. Furthermore, a smaller program is simply a series of constructs (sequence, alternation, iteration, and concurrent) which are also function rules and the mathematics, again, apply. Thus, a group of constructs can be replaced by its surrogate function and, in turn, a group of program can be replaced by its function, etc. until a complete proof for the large system is complete. Axiomatic verification does not scale-up so readily but becomes more complex as programs become larger and more realistic. Its use on only small, classroom-type problems has kept axiomatic verification entrenched in academia resulting in turning off generations of students against program verification.

We recommend the use of functional verification techniques to develop deductive proofs of program correctness because of its scale up. These types of proofs can be learned and applied and result in higher quality software. The mathematics of functional verification are summarized in Appendix A.

Organizations that have substituted functional verification (with team reviews) in place of code inspections have been very successful in producing failure-free software. Programming groups which faithfully practice structured programming, top-down design, and functional verification have had outstanding successes in developing high quality software. A leading example of this is the IBM group in Houston that develops the space shuttle software. The methods this group follows were established by Harlan Mills, one of the authors of this paper. Appendix C provides more details on this project and other successful projects developed using structured programming, top down design with stepwise refinement, and functional verification.

In 1993, even with the extensive knowledge available in the literature about the hazards of debugging and the advantages provided by verification, most software developers still verify their programs by debugging. Observations on the results of debugging indicate that between 1 time

in 5 and 1 time in 10 (say 15% of the time), the result of changing a program, in response to an observed failure, is that a deeper fault is placed in the software which will cause a failure at some future time. This indicates that debugging is most likely the single largest cause of observed production problems. It is clear that any organization that wants to produce quality software should, at least, reduce debugging and preferably eliminate debugging as a means for verifying programs. Observations on the results of replacing debugging by functional verification indicate that developers not only produce software that contains fewer latent operational failures but they also increase their development productivity. These observations indicate that the above referenced table prepared by Dyer and Kouchakdjian can be extended by adding a new row we have labeled "Early Cleanroom". Cleanroom is the name given to the ensemble engineering and management practices created by Mills and SET that rely on the scientific foundation for software development as summarized Appendix A. Cleanroom engineering practices include the replacement of independent debugging by the development team with functional verification.

Development Practices	Development Defects/KLOC	Operational Defects/KLOC	Productivity LOC/Staff Mo
Traditional (unstructured programs, bottom-up design)	50 - 60	15 - 18	unknown or unmeasurable
Modern (structured programs, top-down design, code inspections)	20 - 40	2 - 4	75 - 475
Early Cleanroom (modern plus functional verification and Box Structure design)	< 5	<< 1	> 750

The full impact of the quality improvement is not due to the use of functional verification but it plays an important part. Appendix A provides more data on the disadvantages of using debugging as a means to verify the accuracy of programs.

Cleanroom is a formal and complete software engineering method. We believe that of all existing formal methods for software engineering, only Cleanroom is a complete strategy encompassing specification, development with verification, and statistical certification. Many strategies which claim to be formal methods only contain only one these ingredients. In addition, Cleanroom has a proven track-record of success. Projects both large and small have been successfully completed with excellent results (see the appendix for some specific details). Thus, in Cleanroom, developers have a formal method which is practical to apply, improves quality and productivity, and spans the entire software lifecycle.

The previous discussion on debugging points out how hazardous debugging can be when trying to develop a correct program. As a result the NRC should be extremely skeptical of any

software submitted for licensing where the developers used debugging as a means to support program verification.

3.3 Top-Down Design

During the late 1970s and through the 1980s the leaders of the movement to establish a software engineering practice struggled with how best to develop a top-down design. There were three basic approaches that each have advocates; however, none of the three approaches have proven adequate to reliably guide developers in correct top-down design. It was not until 1989 that Mills defined an approach to top-down design that unified the three previous approaches and created the first engineering method for correct top-down design.

There are three factors that need to be considered in software development (i) the process to follow, (ii) the data to be manipulated, and (iii) the objects in which to group processes and data. Each of the methods suggested during recent years have emphasized one of these three factors with a minor focus on the other two. This is depicted in the following table.

Major Concentration	Method	Innovator
Process	Structured Design Jackson Development Methodology	Yourdon Jackson
Data	Warnier-Orr Information Engineering	Warnier and Orr Martin
Objects	Booch Eiffel Coad and Yourdon	Booch Meyer Coad and Yourdon
Unified	Box Structures	Mills (1989)

Mills has found that only three forms of functions are required when developing software [5]. The three forms are black boxes, state boxes, and clear boxes. The form of these three functions are summarized in the following table.

Form	Mathematical Expression	In English	Comments
Black box	$f: S^* \rightarrow R$	System responses defined in terms of stimuli histories.	Implementation free. Object view.
State box	$g: (S \times T)^* \rightarrow (R \times T)$	System responses and updated state data defined in terms of current stimulus, state data being maintained at this level and stimuli histories for those stimuli that are to maintained at a lower level in the usage hierarchy.	Generalization of state machine that permits data to be migrated through out the usage hierarchy. Data view.
Clear box	$\text{process } (S \times T)^* \rightarrow (R \times T)$	System responses and updated state data defined in terms of an algorithmic process that combines the current stimulus, state data being maintained at this level of the usage hierarchy and references to lower level black boxes.	A program is a process. Process view.

The Box Structures design algorithm unifies these three important views of software, making it possible to fully develop a correct top-down design. In 1989 Mills defined an algorithm for the design of software systems using Box Structures. The algorithm proceeds by developing each software object top-down from a black box, to a state box, and then to a clear box. Clear boxes introduce the need for new objects or black boxes. The process continues until all leaf clear boxes do not introduce the need for lower level objects or black boxes. In addition, Mills specified how to verify each refinement from box to box. This is critically important because if no mistake is made in performing a functional verification then the resulting software will perform exactly as specified. When developing software with the box structures algorithm, correct software and its proof of correctness are constructed in parallel.

Additional details on the mathematics of box structures and the design algorithm are included in Appendix A.

If humans were infallible, then this method would always result in correct software. Unfortunately, humans sometimes are fallible. There are two solutions to human fallibility in preparing proofs.

Independent testing. We can conduct tests, performed by an independent team, to look for failures in the functional proofs. This subject is discussed in more detail in the next subsection.

Formality of proof recording. There is a trade off between the formality of a proof of correctness and the extent to which the proofs are recorded. If every proof is written out and in full detail and the proofs are subjected to several independent peer reviews, the cost of development increases but the probability of any error remaining undetected in the

proofs will approach 0. If, on the other hand, only the more complex proofs are recorded in full detail and only a few proofs are verified outside of the development team, the cost of development declines but the chance that an error will remain undetected increases. Experience indicates that very reliable code can be produced with teams performing proof reviews of every expansion but not fully recording all proofs. For extremely important parts of the software that need to be completely reliable, experience dictates that it is best to record all proofs. Our experience is that this practice produces failure-free software.

Since the shutdown software must be correct, it is recommended that all proofs for the shutdown software be formally recorded and extensively reviewed to ensure the proofs are performed correctly.

Another important derivative of understanding the mathematics of box structures is that it is now known that the form of the function used to specify software behavior should be a black box. This is because a black box function is implementation free and is understandable by anyone who understands the stimuli the software receives and the responses it issues. Thus, ensuring the effectiveness of independent reviews of the specification and proofs of correctness.

3.4 Software Testing and Certification

In the very early days of software development, there was no testing theory for software. Each software developer did what they thought to be best.

Then in the 1970s and 1980s a great deal of attention was focused on structural-based testing. The idea behind structural testing was to develop methods for defining test cases from a study of the program's structure. The reasoning went something like this: since software failures are the result of bugs in the code, testing should exercise as much of the code as possible to ensure the bugs have been removed. Thus, methods such as functional and statement coverage testing received a great deal of attention in the literature. In practice, these structural techniques were well received because they gave software testers a goal to pursue, i.e., 100% coverage of the code. Testers were comfortable with this approach because they thought they always knew where they stood and how much they had to do to be finished.

As the experience base with structural testing grew it became evident that many errors were still present in software that had received 100% coverage. Even though it is true that the bug in the code is the *cause* of the error, the execution of the code has little to do with the *appearance* of the error. Instead, errors are manifested in the *usage* of the software. The same bug may cause a failure under one usage circumstance and not another. In fact, we argue that the appearance of a failure is completely dependent on the usage of the software. If a function is not used, it will not fail no matter how many bugs are present. Furthermore, even novice programmers have experienced the surprise and frustration of users encountering failures that somehow managed to get past, what seemed to be, thorough testing. Again, the reason for the appearance of such failures is not because the programmer failed to execute the faulty section of code, it is because

it was executed with a different sequence of inputs (i.e., a different usage history). Thus, the reason that structural testing fails to uncover many software errors is because it does not address the true cause of software failure: usage.

In the 1980s, it began to be recognized that usage-based testing was a better way to test software since structural-based testing is inferior to usage-based testing in satisfying testing objectives. Usage-based testing is based on preparing a sample of test cases that reflect the uses that some typical user may have supplied to the software during some period of time. Tests of software should be conducted for two basic reasons as discussed below.

1. Using testing to identify and then fix failures to increase the reliability of software.

Usage-based testing is 30 times more effective than structural-based testing at finding failures that users are likely to encounter so structural-based testing is not cost effective at increasing MTTF for software prior to release. See Appendix B for the details of this estimate.

2. Using testing to certify the correctness of software.

The goal of certification testing is to develop an estimate of the reliability of the software. If we generate test cases by either following some path through the software or by representing a possible use of the software by some user, the total number of possible test cases is very large. In the order of trillions^{billions}. To see how large the number can be, consider a square root program for a 64 bit processor. The number of possible test cases is nearly 10^{20} . Since the number of paths is so large, it is not possible to exhaustively test the software. Therefore, a sample must be used to estimate software reliability. Since there is no way to characterize the distribution that represents the combinations of paths through the software and state data values, the only statement that structural testing allows about software reliability are anecdotal arguments. Since 1990 it has become possible to characterize the probability density function for how a class of users will use a software system. We now know that a Markov chain can be used to represent the required probability density function. As a result, an organization that applies usage-based testing can develop a proper random, representative sample of uses. The uses that comprise the sample can be executed on the software and sample statistics can be used to develop estimates of the entire population of uses.

It has been known since the mid 1980s that usage-based testing was the way to certify software but usage of a software system is not an easy entity to deal with. It is a complex process of applying sequences of stimuli and receiving responses. In most cases, the application of stimuli have multiple dependencies associated with them. For example, the frequency of stimulus A for a software system may depend on the occurrence of a prior stimulus in the same execution or even in a past execution of the software. Thus, the application of stimuli requires that the entire past usage history be exposed and analyzed in order to adequately simulate usage patterns of the

software.

It was not until 1990 that all the critical knowledge was in place to permit people to apply usage testing to software certification and it was not until 1992 that sufficient projects have been conducted to fully define the engineering practices to support usage-based testing. Subsequent research by SET (funded, in part, by the Department of Defense) has perfected these methods, and it is now possible to concisely model large and complex software systems using Markov models. The Markov model has two main components. First, a *structural* component models the pertinent usage history information and its relationship to the input domain. This component makes possible effective and efficient reviews of the test plan and is the basis for automatic generation of test data. The structural component is modeled as the states of a Markov chain. Each state represents an event in the stimulus history which impacts ensuing use of the software. In this manner, any and all pertinent usage history is "remembered" in the *finite* state set. States are connected via directed arcs labeled with stimuli from the software's input domain. There is an initial state of the system corresponding to "no stimulus yet received", i.e., some non-invoked or startup state and a final state representing software termination or power down. A path, in the form of a connected state/stimulus sequence from the initial state to the final state constitutes a use of the software. Thus, we can build a finite model which effectively generates the infinite language which is use of the software. This model has made an enormous impact on the ability to effectively test and accurately certify software quality. Aspects of testing that were never a part of traditional statistical tests are now possible because of the model. For example, an integral part of the Markov model is the direct encapsulation of stimulus history (i.e., the software's internally saved-state) which creates the basis for meaningful coverage criteria. We can measure important aspects of software usage such as whether or not all the stimuli have been applied from all possible state configurations; a strong statement about the completeness of any testing strategy.

The second component of the Markov model is the statistical component, which makes possible a scientific certification. The *statistics* of the usage model are in the form of probability distributions over the transitions (arcs) from each state. Whereas the structural of the model is constructed using knowledge of the software's specification, information about appropriate probability distributions comes from other sources, such as captured usage data from prototypes, simulators, or prior software versions. This data is supplemented as necessary via interviews of potential customers to gain intuition into expected usage patterns. The Markov model exactly pinpoints the probabilities necessary to complete the model, i.e., the probability that a given stimulus is applied from a particular state configuration. This simple identification of exactly the item needed has simplified the task of estimating probabilities a great deal.

The statistical component is an important aspect for more than just the obvious reasons. Indeed, safety-critical software in nuclear power plants or on space vessels is well-suited for statistical certification. However, all software developers, managers, and users benefit from the opportunity to measure and, thereby, increase the quality of their products. Statistical testing is a well-controlled experiment, and therefore, costs and risks associated with it are well-controlled.

Scientific measurements are made with each test case applied so that the progress and cost of testing are always known.

The creation of this *usage model* is an engineering-based process deeply rooted in sound mathematical and statistical theory. However, it does not require mathematicians or statisticians to implement the techniques. Properly trained engineers have already succeeded in several projects and many more are in progress. As the science of usage-based statistical testing progresses, new theories, tools, and techniques are emerging; making the regime easier to implement and increasing its benefit to software development organizations.

The data produced during a usage-based software test represents a powerful medium with which to compute estimates of a software product's reliability. There are two situations in which computation plays an important role in software performance evaluation.

First, when failures are observed during testing engineers need to analyze the impact of the failures on the quality of the software. It is now obvious that the software is incorrect and must be re-engineered but it is informative to determine the likelihood, given the statistics of the usage model, of the failure appearing in real use. The information could be used to determine when to repair the software fault or to estimate the quality of the software as is, i.e., without repairing the fault.

In the case of failures occurring, a second Markov chain is created to model the occurrence of the failure. This *testing chain* is constructed given an entire test history (test cases executed) of a software product. This chain is the basis for reliability estimation. Because it is a well-defined stochastic process itself, the computations are made using the theory of Markov chains and required no further assumptions. The details of the testing chain is included in appendix A.

Second, when failures are not observed, the data indicates that the software is correct. However, since exhaustive testing cannot be performed in order to be guaranteed that it is, indeed, correct; an estimate of the confidence in the correctness is necessary. The data available for analysis is the actual test data applied to the software. One would expect that more testing performed would indicate that the confidence in software correctness is higher, however, many reliability models are unable to capture this notion. There is another important fact besides more data, that is, whether or not the data is *statistically typical* of real use. We are able to account for this because of the Markov model. "Real use" for us is well-defined by the Markov model. Therefore, avoiding any further assumptions, we can compute the confidence in the correctness of the software based on the statistics of the sequences generated compared to the model itself.

For both cases, the fact that we have a Markov model, a formal mathematical system from which computation is readily possible, as a basis from which to estimate is of great value. The appendix shows, in detail, the mathematics that support computation about software quality. These results allow the NRC to expect a manufacturer to supply a valid estimate of software reliability with their licensing application through usage-based testing to certify software.

3.5 Software Specification

When developing software, two important transformations are made so that the software will meet the mission assigned to it. The first transformation is the preparation of a set of specifications which define desired software behavior. The second transformation is the development of software that behaves in the manner described in the specifications.

Software developers and researchers have been struggling with how best to perform both transformations. In previous sections, we have been addressing the second transformation. The science base for software development, completed by Mills in 1990, makes it possible for disciplined teams of engineers to accurately and efficiently perform the second transformation. However, the first transformation is the more difficult transformation. To develop specifications it is necessary to (1) fully understand the mission being assigned to the software including all its ramifications, (2) invent stimuli and responses that provide convenient and foolproof means for software users to interact with the software, (3) invent the behavior required of the software to generate the correct response for all combinations of previously received stimuli, both correct and incorrect, and (4) document the results of the inventions in a form which sponsors, developers, and certifiers can understand. Sponsors must understand the specifications to ensure the software behavior, as defined, is what they require, while developers and certifiers must understand the desired software behavior so they can carry out their mission.

A great deal of guidance has been issued on how to prepare specifications. It has been known for a long time that a specification should satisfy the following requirements.

- the function that defines the desired behavior must be included in the specification
- the form of the function should be understandable to all concerned
- the stimuli and responses must be fully and completely defined

A great deal of effort has been expended in searching for the best form in which to define the desired behavior. The following table defines some of the forms that received significant attention.

Method To Model Behavior	Comments
Graphical representations of behavior in terms using various sorts of block diagrams (i.e., data flow diagrams, IDEF ,...)	Block diagrams have proven useful for communicating concepts and for outlining flows. Block diagrams have proven to provide an inadequate foundation for design since essential details of behavior need to be omitted to fit the model.
State machines represented either as diagrams or equations	State machines can provide an accurate representation of behavior but they suffer from two difficulties: (1) the state data used are in reality an internal invention so communications are difficult and (2) a state machine requires all state data to be maintained at the highest level in the usage hierarchy. We know that in a software system state data should be migrated to the lowest practical level in the hierarchy. The state box, which is used in Box Structures mathematics, is a generalization of the state machine invented by Mills in the late 1970s to solve this problem.
Process definitions.	Process definitions are the natural way to describe a current process. The difficulty with using a process description in a specification is that the specification then defines what is to be done by describing how to do it. This makes it very difficult to understand the essence of what needs to be done.
Black Box function	We now know since Mills work that the best way to define a function is as a Black Box function. We know that every object has a black box function. Every other description of behavior can be converted into a black box function. Black box functions are ideal for specifications since they define what has to be done in terms completely external to the thing being specified. As a result black box functions describe behavior in a manner that is understandable by anyone who understands the stimuli and the responses.

For reasons explained earlier, Cleanroom engineers utilize black box functions in specifications. A Cleanroom specification is contained in 6 parts as follows.

Mission	A precise statement of the requirements
User's Reference Manual	A definition of the stimuli and responses invented so the software can fulfill its mission.
Software Function	A black box function that defines software responses in terms of stimuli histories (ie, in an implementation-free manner)
Specification Verification	A rigorous argument that the software as defined will meet its defined mission.
Software Usage Profile	A Markov Model stating the probability of moving from each usage state to all other usage states.
Construction Plan	A plan for building the software in a series of increments such that each accumulation of increments is executable by user stimuli.

Box structures mathematics play an important part in preparing a specification just like they do in designing software. As stated above, a specification team is often given a process description for portions of a system that is to be automated. Using the same mathematical relationships to verify the correctness of the design in reverse, an engineer can abstract the black box function that defines the behavior provided by the clear box. This abstraction method presents a very valuable tool to the specification engineer.

Another important ingredient of the specification is the Markov model that describes behavior from the vantage point of an external observer. We are now experiencing that the Markov model and the black box function provide complementary views of behavior that help define the best possible specification.

We believe validating the specifications will present the most serious challenge relative to verifying the correctness of the shutdown system. The specification verification task includes two subtasks. The first is a specification of the behavior desired for the shutdown channel. The second part of the specification is a definition of the behavior that has been allocated to each of the components in the shutdown channel. The verification of the allocation will not be difficult since all we need to show is that the composite behavior of a sequence of black boxes provides the behavior specified for the system as a whole. The critical issue is to ensure that the specification for the entire channel is correct in that it does not omit any safety related issue. It is always difficult to show that every possible issue has been considered. It is much easier to show that all considered issues have been considered correctly. We assume the engineers and scientists who understand the operation of power plants are in a position to develop a good

specification of the desired system behavior.

The specification of the shutdown system must contain a solution to the common mode failure issue. The common mode failure issue can be summarized as follows.

The concern is that a specification or an implementation failure may result in a common mode or common cause failure of redundant equipment.

The two solutions to the common mode failure problem are (1) to install only quality products that contain no specification or implementation failures and (2) to install diverse products so that the probability of common mode failures are reduced. It is at the specification stage that the solutions to the common mode failure problem need to be addressed. In terms of software, the procedures for development of the implementation address the quality issue. Diverse software components can be developed from a common function since there are an infinite number of correct rules for any given function.

Professor Leveson, who has testified earlier and will be testifying again, has been leading an effort to get software developers to focus on the importance of developing correct specifications for safety critical applications. In our view she is correctly concerned about the importance of searching for situations that are easily omitted from a specification that can cause a safety related accident. She wants us to minimize the possibility that one or more possible factors are left out.

We feel that her message about the importance of defining a correct specification is often getting lost because of a lack of a clear distinction between specification failures and implementation failures. Appendix D to this paper contains an analysis of her testimony to this committee on August 21, 1992. This analysis highlights the need to separate the issue of specification and implementation correctness.

The reason the distinction is important is because the action that must be taken is different in each case. If software is regarded to be correct when it exhibits behavior exactly equivalent to its specification for all stimuli within the legal domain and it behaves as required for all stimuli outside of the legal domain then the concern of minimizing the possibility of safety related failures becomes a specification issue. This is exactly where it belongs.

The methods discussed in this paper provide the means to develop correct software. The methods discussed in this section help to define a clear, unambiguous specification so all interested parties to the specification can reason about the proposed solution. The specification should be complete and consistent before implementation is started. It should be pointed out that typically in software projects there is not a clear distinction between specifications and implementation. There is another aspect of craft-based software development that has been corrected in Cleanroom.

We believe that using a Cleanroom specification as a starting point will promote a complete

safety analysis which can be conducted by the engineers who understand the power plant domain.

3.6 Management

The development of software not only requires a theory for how to design software but it is also necessitates a method of managing a software project. Most software projects are conducted by teams of people whose work must be coordinated. The issues relative to managing software projects include:

Intellectual Control Software development is a process. In order to maintain the project in a state of intellectual control, the process to be followed must be defined and communicated so it can be easily followed by all project participants. The process must define the control and data flow among processes and specify the manner in which each activity is to be performed. The control flow is quite complex since there are many activities being performed in parallel, at least one for each member of the team. Subsequent tasks should not start until all prior tasks are complete. The next process or activity to be performed is not known in many cases until the nature of the work product of the prior activity is appraised. Until the 1990's all these important complexities were ignored when preparing models of software projects. Consequently projects were consistently out-of-control. In the 1991/92 time frame, progress has been made in developing and using realistic models of the software development process. Some projects have now been organized using these findings. To date such process models have only been developed for projects using Cleanroom practices and processes.

Teams Cleanroom development practices utilize three teams: the specification team, the development team, and the certification team. The specification team is responsible for preparing and maintaining the software specification. The development team is responsible for refining the function contained in the specification into a rule for the function that will execute on the target processor(s) with the required degree of efficiency. They are also responsible for verifying that the rule is correct by applying functional verification techniques. The certification team is responsible for testing the software product and certifying its correctness.

Completion Conditions and Proof Reviews Software development is a complex, creative activity. Humans need help and support in performing these activities. Well trained teams provide the best means to support humans in developing software. We recommend that teams be accountable for results. Completion conditions can be developed for each activity a team or team member must perform. Teams can then complete a "completion conditions form" when they believe the work has been executed satisfactorily. In this way, the team and the organization's management can be sure the team believes they have fully and completely finished the required work. Since programs are mathematical objects, we recommend that all software refinement activities be followed by verification with a team review of the proof argument. In this way the individual developer and all

of his associates become convinced of the correctness or lack of correctness of each refinement.

Incremental Development Many software objects are quite large. In order to develop the software using statistical process control it has been found that it is desirable to decompose the development effort into smaller units, called increments. Increments should be chosen so that they can be formed into a sequence such that each accumulation, including the first, can be executed with user supplied stimuli and the results appraised with user observable responses. The development team produces the software increment by increment. When the development team completes the current increment it is turned over to the certification team. The certification team then assembles the new increment with previous increments. They test the new accumulation using test scenarios which have been generated for the accumulation from a Markov chain developed for the accumulation. Increments are chosen to be fairly small. In this way, a development team will complete an increment every 3 to 5 weeks so that project management obtains a continuous measure of software quality. The small increments enable management to quickly take appropriate action if quality begins to deviate from target levels. This development pattern also assures both organization and customer management that there will be no last minute surprises due to the integration of various software parts.

The Cleanroom management practices are designed to amplify the engineering practices so the entire project can be efficiently performed under the intellectual control of its participants.

4.0 References

- [1] D. W. Dijkstra, "Structured programming", Software Engineering Techniques, J. Burton and B. Randall eds., NATO Science Commission, 1969.
- [2] Dyer and Kouchakdjian, "Correctness verification", Information and Software Technology, October 1990.
- [3] C. A. R. Hoare, An axiomatic approach to computer programming, Communications of the ACM, Vol. 12, no. 10, October 1969.
- [4] H. D. Mills, Software Productivity, Little, Brown and Company, 1983.
- [5] H. D. Mills, "Stepwise refinement and verification in box-structured systems," IEEE Computer, June 1988, pp 23-36.
- [6] N. Wirth, "Program development by stepwise refinement", CACM 14, 4, April 1971, pp 221-227

Appendix A

Cleanroom Engineering: A Brief Overview

The purpose of this note is to provide a brief overview of the Cleanroom Engineering practices.

1.0 What is Cleanroom Engineering of Software?

The Cleanroom Engineering process develops software of certified correctness under statistical quality control in a pipeline of increments, with box structured design and functional verification but no program debugging permitted before independent statistical usage testing of the increments. It provides rigorous methods for software specification, implementation, and certification that are capable of producing low or zero defect software of arbitrary size and complexity. Box structured design is based on a Parnas usage hierarchy of modules. Such modules, also known as data abstractions or objects, are described by a set of operations that may define and access internally stored data. Functional verification is based on the fact that any program or program part is a rule for a mathematical function. It may not be the function desired, but it is a function.

The term Cleanroom is taken from the hardware industry to mean an emphasis on preventing errors, rather than allowing errors to appear and removing them later (of course any errors introduced should be removed). Cleanroom Software Engineering involves rigorous methods that enable greater control over both product and process. The Cleanroom process not only produces software of high correctness and high performance, but does so while yielding high productivity and schedule correctness. The intellectual control provided by the rigorous Cleanroom process allows both technical and management control.

Cleanroom Engineering achieves statistical quality control over software development by strictly separating the design process from the testing process in a pipeline of incremental software development. There are three major engineering activities in the process [2, 6].

First, a specification team creates an incremental specification that defines a pipeline of software increments that accumulate into the final software product, which specification includes the statistics of its use as well as its function and performance requirements;

Second, a development team designs and codes increments specified using box structured design and functional verification of each increment, with delivery to certification with no debugging beforehand, and provides subsequent correction for any failures that may be uncovered during certification;

Third, a certification team uses statistical testing and analysis for the certification of the software correctness to the usage specification, notification to designers of any failures discovered during certification, and subsequent recertification as failures are corrected.

As noted, there is an explicit feedback process between certification and development on any failures found in statistical usage testing. This feedback process provides an objective measure of the correctness of the software as it matures in the development pipeline. It does, indeed, provide a statistical quality control process for software development that has not been available in this first human generation of trial and error programming.

1.1 Cleanroom Software Engineering Methods

Cleanroom Engineering provides a set of rigorous methods for software development under statistical quality control, based on sound mathematical and statistical principles. While millions of people are involved in software, most of them regard software development as an intuitive, heuristic activity. They do not imagine software engineering as a mathematically based subject with complete rigor being possible. But software engineering should be and can be a mathematics based activity. When mathematical rigor is applied both quality and productivity increase. Nor can they imagine software engineering based on statistics since computers are completely deterministic in behavior. And yet the usage of software is statistical in nature.

Software is a human generation old, while mathematics is many human generations old. Although not understood early or widely, software has direct mathematics foundations because of the very deterministic behavior of computers. A computer program is a rule for a mathematical function, mapping all possible initial states into final states. Such functions are very complex compared to functions in physical science and engineering, and traditional mathematical notation is very insufficient. But sufficient mathematical notation is emerging in computer science and software engineering for dealing with the syntax and semantics of programs and their functions.

As an example of deep and useful mathematics, place notation and long division moved arithmetic from intuition and heuristics to rigorous methods a thousand years ago in the western world. As a result, school children today can out perform Archimedes and Euclid in arithmetic. Similar movement is possible in software today.

Statistics is another subject of longer professional development than software. But only a hundred years ago, statistics was intuitive and heuristic, even though rigorous arithmetic was used in creating sums and averages. Yet in this time, statistics has become a rigorous, mathematics based subject, often finding counter-intuitive results using statistics in specific topics. The application of statistics makes it possible for software developers to predict with confidence the quality level of the software when it is fully developed quite early in the

development life cycle.

Cleanroom Engineering not only puts software development under statistical quality control, but takes out debugging from the list of developer activities, instead using mathematical reasoning before independent testing and certification. Just as typists looked at the keys when typewriters first came out, programmers have felt the need to debug programs in this first human generation of programming. But while counter-intuitive at the time, typists went to touch typing with both higher productivity and fewer errors. In the same way, well educated software engineers can create software with no execution or debugging before it is tested by independent test and certification engineers with the product having higher productivity and much greater quality.

1.2 Dealing with Human Fallibility

Humans are fallible, even in using sound mathematical processes in functional verification, so finding software failures are possible during the certification process. But there is a surprising power and synergism between functional verification and statistical usage testing [6]. First, as already noted, functional verification can be scaled up for high productivity and still leave no more errors than heuristic programming often leaves after unit and system testing combined. Second, it turns out that the mathematical errors left are much easier to find and fix during testing than errors left behind in debugging, measured at a factor of five in practice [6]. Mathematical errors usually turn out to be simple oversights in the software, whereas errors left behind or introduced in debugging are usually deeper in logic or wider in system scope than those fixed. As a result, statistical usage testing not only provides a formal, objective basis for the certification of correctness under use, but also uncovers the errors of mathematical fallibility with remarkable efficiency.

In Cleanroom Engineering, a major discovery is the ability of well educated and motivated people to create nearly defect free software before any execution or debugging, with less than five defects per thousand lines of code. Such code is ready for usage testing and certification with no unit debugging by the designers. In this first human generation of software development, it has been counter-intuitive to expect software with so few defects at the outset. Typical heuristic programming leaves fifty defects per thousand lines of code, then reduces that number to five or less by debugging.

The mathematical foundations for Cleanroom Engineering come from the deterministic nature of computers themselves. As noted, a computer program is no more and no less than a rule for a mathematical function [3, 4]. Such a function need not be numerical, of course, and most programs do not define numerical functions. But for every legal input a program directs the computer to produce a unique output, whether correct as specified or not. And the set of all such input, output pairs is a mathematical function. A more intuitive way to view a program in this first generation is as a set of instructions for specific executions with specific

input data. While correct, this view misses a point of reusing well known and tested mathematical ideas, regarding computer programming as new and private art rather than more mature and public engineering.

With these mathematical foundations, software development becomes a process of constructing rules for functions that meet required specifications, which needs not be a trial and error programming process. The functional semantics of a structured programming language can be expressed in an algebra of functions with function operations corresponding to program sequence, alternation, and iteration [3]. The systematic top down development of programs is mirrored in describing function rules in terms of algebraic operations among simpler functions, and their rules in terms of still simpler functions until the rules of the programming language are reached. It is a new mental base for most programmers to consider the complete functions needed, top down, rather than computer executions for specific data.

2.0 Box Structured Software System Design

Box structured design is based on a **Parnas** usage hierarchy of modules [9, 10]. Such modules, also known as data abstractions or objects, are described by a set of operations that may define and access internally stored data. In Ada, such modules are defined as **packages**, with operations defined by the calls of the procedures and functions of the packages, and internal data declared in the package.

Stacks, queues, and sequential or random access files provide simple examples of such modules or packages. Part of their discipline is that internally stored data cannot be accessed or altered in any way except through the explicit operations of the package. It is critical in box structured design to recognize that packages exist at every level from complete systems to individual program variables. It is also critical to recognize that a verifiable design must deal with a usage hierarchy rather than a parts hierarchy in its structure. A program that stores no data between invocations can be described in terms of a parts hierarchy of its smaller and smaller parts, because any use depends only on data supplied it on its call with no dependence on previous calls. But a specific realization of a package, say a queue, will depend not only on the present call and data supplied it, but also on previous calls and data supplied then.

The parts hierarchy of a structured program identifies every sequence, alternation, and iteration (say every begin-end, if-then-else, while-loop) at every level. It turns out that the usage hierarchy of a system of packages (say an object oriented design with all objects identified) also identifies every call (use) of every operation of every package. The semantics of the structured program are defined by a mathematical function for each sequence, alternation, and iteration in the parts hierarchy. That doesn't quite work for the operations of packages because of usage history dependencies. But there is a simple extension for packages that does work. It is to model the behavior of a package as a **state machine**, with its calls of its several operations as inputs to the common state machine. Then the semantics of such a

package is defined by the **transition function** of its state machine (with an initial state). When the operations are defined by structured programs, the semantics of packages becomes a simple extension of the semantics of structured programs.

2.1 The Basis for Box Structured Design

While theoretically straightforward, the practical design of systems of Parnas modules [10, 11] (object oriented systems) in usage hierarchies can seem quite complex on first exposure. It seems much simpler to outline such designs in parts hierarchies and structures, for example in data flow diagrams, without distinguishing between separate usages of the same module. While that may seem simpler at the moment, such design outlines are incomplete and often lead to faulty completions at the detailed programming levels. In spite of their common use in this first human generation of system design, data flow diagrams should only be used within rigorous design methods rather than leaving critical requirements to details with incomplete specifications.

In order to create and control such designs based on usage hierarchies in more practical ways, their box structures provide standard, finer grained sub-descriptions for any package of three forms, namely as **black boxes**, as **state boxes**, and as **clear boxes**, defined as follows [5, 7, 8].

Black Box: External view of a Parnas package, describing its behavior as a mathematical function from historical sequences of stimuli to its next response.

State Box: Intermediate view of a Parnas package, describing its behavior by use of an internal state and internal black box with a mathematical function from historical sequences of stimuli and states to its next response and state, and an initial internal state.

Clear Box: Internal view of a Parnas package, describing the internal black box of its state box in a usage control structure of other Parnas packages; such a control structure may define sequential or concurrent use of the other packages.

Box structures enforce completeness and precision in design of software systems as usage hierarchies of Parnas packages. Such completeness and precision lead to pleasant surprises in human capabilities in software engineering and development. The surprises are in capabilities to move from system specifications to design in programs without the need for unit/package testing and debugging before delivery to system usage testing. In this first generation of software development, it has been widely assumed that trial and error programming, unit testing and debugging were necessary. But well educated, well motivated software professionals are, indeed, capable of developing software systems of arbitrary size and complexity without program debugging before system usage testing [2].

3.0 Stepwise Refinement and Functional Verification of Software

Once the design is complete, the clear box at each level is expanded to code to fully implement the defined function rule for the black box function at that level by stepwise refinement, as introduced by Wirth [12]. Following each expansion functional verification is used help structure a proof that the expansion correctly implements the specification. The nature of the proof revolves around the fact that a program is a rule for a function and the specification for the program is a relation or function. What must be shown in the proof is that the rule (the program) correctly implements the relation or function (the specification) for the full range of the specification and no more. Linger, Mills and Witt [3] have developed a correctness theorem which defines what must be shown to prove that a program is equivalent to its specification for each of the structured programming language constructs. The proof strategy is subdivided into small parts which easily accumulate into a proof for a large program. Experience indicates that people are able to master these ideas and construct proof arguments for very large software systems.

The development team expands each clear box in the usage hierarchy into the selected target code using stepwise refinement and functional verification. As the development team designs and implements the software, it is held collectively responsible for the quality of the software.

In describing the activities of software development, no mention is made of testing or even of compilation. The Cleanroom development team does not test or even compile. They use mathematical proofs (functional verification) to demonstrate the correctness of programming units. Testing and measuring failures by program execution is the responsibility of the certification team.

3.1 The Mathematical Basis for Functional Verification

Any program or program part is a **rule** for a **mathematical function**. It may not be the function desired, but it is a function. In structured programs, the rules are direct in form, building program rules out of just two function building operations, first, **function composition** which corresponds to sequential execution of program parts, and second, **disjoint function union** which corresponds to alternative execution of one program part or another, as in if or case structures. Program iteration uses no more than these two operations together, and function recursion provides a useful view of an iteration process.

As noted, any program part or total program defines a single, possibly complex function. The function is seldom a numerical function in classical terms. Even numerical programs must deal with finite sets of numbers in which overflow and roundoffs depart from classical number systems. Given the text or name of a program or program part in whatever language, say a program in Ada

```
Alpha =      with text_io;
              procedure Beta
              is
              ...
              begin
              ...
              end Beta;
```

the program function will be denoted by brackets [,] around the name or text, as

```
[Alpha] = [with text_io;
            procedure Beta
            is
            ...
            begin
            ...
            end Beta;]
```

In this case [Alpha] is a set of ordered pairs

[Alpha] = { <X, Y> | Given initial state X, Alpha will produce
final state Y }

The function [Alpha] is determined by Ada text, but is independent of the language Ada. The same function can be defined in Fortran text, COBOL text, etc.

3.2 Functional Verification of Program Parts

From programs to program parts, starting with simple assignment statements, such as

```
x := y;
```

in Ada, the program part function

```
[x := y;]
```

takes its initial data state to its final data state. If legal, it will change the value of x in the final state to the value of y in the initial state and change no other values of variables in the initial state. If illegal, the final state may be quite different than the initial state, possibly with both x and y disappearing, as well as other variables, in terminating the entire program execution. So assignment statements have simple function parts when legal, but possible more complex function parts when illegal. In summary, the function [x := y] is a set of ordered pairs with second members determined uniquely by the first members

$$[x := y;] = \{ \langle \langle x, y, \dots \rangle, \langle y, y, \dots \rangle \rangle \mid x := y; \text{ is legal} \} \\ \cup \{ \langle \langle x, y, \dots \rangle, \langle ??? \rangle \rangle \mid x := y; \text{ is illegal} \}$$

where ??? will be determined by other aspects of the initial state. Illegal situations will be suppressed in what follows for sake of time. In more direct function notation, dealing only with the legal situation,

$$[x := y;](\langle x, y, \dots \rangle) = \langle y, y, \dots \rangle$$

in which the function **argument** $\langle x, y, \dots \rangle$ produces the function **value** $\langle y, y, \dots \rangle$.

Next, for a sequence of statements, such as

$x := y; y := z; z := x;$

in Ada, the part function

$$[x := y; y := z; z := x;]$$

will alter values of x, y, z as a composition of the three individual assignment functions

$$[x := y;] * [y := z;] * [z := x;].$$

That is, beginning with an initial state as argument, the first assignment function gives a new state as value

$$[x := y;](\langle x, y, z, \dots \rangle) = \langle y, y, z, \dots \rangle,$$

the second assignment function uses this value as an argument

$$[y := z;](\langle y, y, z, \dots \rangle) = \langle y, z, z, \dots \rangle,$$

and the third assignment function uses this last value as argument

$$[z := x;](\langle y, z, z, \dots \rangle) = \langle y, z, y, \dots \rangle.$$

That is, the composition function is a nested set of simpler functions that evaluate as

$$\begin{aligned} & ([x := y;] * [y := z;] * [z := x;])(\langle x, y, z, \dots \rangle) \\ &= [z := x;]([y := z;]([x := y;](\langle x, y, z, \dots \rangle))) \\ &= [z := x;]([y := z;](\langle y, y, z, \dots \rangle)) \\ &= [z := x;](\langle y, z, z, \dots \rangle) \\ &= \langle y, z, y, \dots \rangle \end{aligned}$$

as worked out just above. In summary, this composition function will interchange the values of y and z and leave x with the initial value of y, not changing any other data in the initial state.

Finally, for an alternation statement, such as

if $x > y$ then $y := z$ else $x := z$ end if;

in Ada, the part function will execute either the then part or else part, so that

[if $x > y$ then $y := z$; else $x := z$; end if;]
= $(x > y \rightarrow [y := z;] \mid x \leq y \rightarrow [x := z;])$
= $[y := z; \mid x > y] \cup [x := z; \mid x \leq y]$

where the expression $[y := z; \mid x > y]$ means the function $[y := z;]$ with its domain restricted to the condition $x > y$. That is, the part function is a union of disjoint functions.

4.0 Software Engineering under Statistical Quality Control

The statistical foundations for Cleanroom Engineering come from adding usage statistics to software specifications, along with function and performance requirements [1, 6]. Such usage statistics provide a basis for measuring the correctness of the software during its development, and thereby measuring the quality of the design in meeting functional and performance requirements. A more usual way to view development in this first generation is as a difficult to predict art form. Software with no known errors at delivery frequently experiences many failures in actual usage.

Cleanroom statistical certification of software involves, first, the **specification of usage statistics** in addition to function and performance specifications. Such usage statistics provide a basis for assessing the correctness of the software being tested under expected use. Because the usage statistics are computed from a well-defined stochastic process, we have created a solid basis from which to compute estimate of the correctness of the software

As each specified increment is completed by the designers, it is delivered to the certifiers, who combine it with preceding increments, for testing based on usage statistics. As noted, the Cleanroom architecture must define a sequence of nested increments which are to be executed exclusively by user commands as they accumulate into the entire system required. Each subsequence represents a subsystem complete in itself, even though not all the user function may be provided in it. For each subsystem, a certified correctness is defined from the usage testing and failures discovered, if any.

It is characteristic that each increment goes through a maturation during the testing, becoming

more reliable from corrections required for failures found, serving then as a stable base as later increments are delivered and integrated to the developing system. For example, the HH60 flight control program had three increments [1] of over 10 KLOC each. Increment 1 code required 27 corrections for failures discovered in its first appearance in increment 1 testing, but then only 1 correction during increment 1/2 testing, and 2 corrections during increment 1/2/3 testing. Code in increment 2 required 20 corrections during its first appearance in increment 1/2 testing, and only 5 corrections during increment 1/2/3 testing. Increment 3 code required 21 corrections on its first appearance in increment 1/2/3 testing. In this case, 76 corrections were required in a system of over 30 KLOC, under 2.5 corrections per KLOC for verified and inspected code, with no previous execution or debugging.

In the certification process, it is not only important to observe failures in execution, but also to record the location of the failure within the statistically generated inputs. Failures in one sequence may or may not have the same statistics as a failure in another sequence. Thus, each failure is identified according to its location within the usage profile. Such test data must be developed to represent the sequential usage of the software by users, which, of course, will account for previous outputs seen by the users and what needs the users will have in various circumstances. The state of mind of a user and the current need can be represented by a stochastic process determined by a state machine whose present state is defined by previous inputs/outputs and a statistical model that provides the next input based on that present state.

As an example, consider the simple selection menu pictured in Fig. 1. The input domain consists of the up arrow key and the down arrow key, which move the cursor to the desired menu item, and the enter key which selects the item. The cursor moves from one item to the next and wraps from top to bottom on an up arrow and from bottom to top on a down arrow. The first item, Select Project, is used to define a project (the semantics of which are not described here for simplicity). The project name then appears in the upper right corner of the screen. Once a project is defined the next three items, Enter Data, Analyze Data, and Print Report, can be selected to perform their respective functions. If no project is defined, selecting these items gives no response.

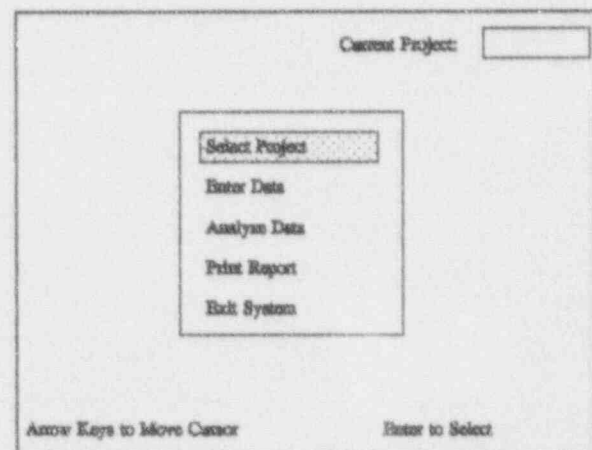


Figure 1: An Example Software System

The first step in defining a usage model is to identify each event of the usage history which impacts user-perceived software behavior (i.e., the usage states). The location of the cursor, which is arrived at through successive application of the arrow keys, is one such item because of the effect that it has on cursor movement. For example, if the cursor is on any of the first

four lines, a down arrow causes the cursor to move down one position while on the fifth line, a down arrow causes the cursor to wrap to the top line. Another event which impacts usage is whether or not a project has been defined. If one has, then the data, analyze, and print options are available, if not, only the select and exit options will function.

Thus, the operational (or usage) state of this example is made up of two usage variables: (i) *cursor location* (which is abbreviated CL and takes on values Sel, Ent, Anl, Prt, or Ext for each respective menu item) because it affects the operation of the arrow keys and (ii) *project defined* (which is abbreviated PD and takes on the values "Yes" or "No") because the operation of the enter key at various cursor locations depends on its value.

The usage states and stimuli are organized as a state transition diagram (STD) that defines legal stimuli (and state) sequences for the software. The transition diagram appears in Fig. 2 for the example menu. The function of the menu items are represented by abstract states and stimuli in order to maintain a concise example.

This STD defines all possible input sequences for the software in a formal, concise model. A path, or connected state/arc sequence, from the initial "Uninvoked" state to the final "Terminate" state represents a single execution of the software. Since loops and cycles exist in the STD, an infinite number of such sequences is possible. A sample stimulus sequence from the STD of Fig. 2 is: **invoke** ↵ **define** ↓ ↓ ↵ **exit** ↑ ↵. The stimuli sequences represent executions of the software and can be used as test cases when supplemented with the expected outcomes as defined by the software's specification.

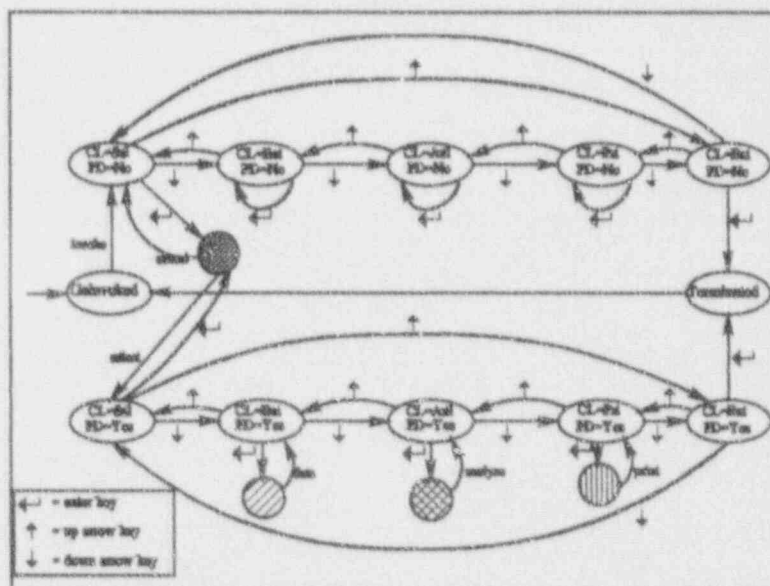


Figure 2: A State Transition Diagram for the Example Selection Menu with No Saved-State

In this example, we note that no prior executions of the software needed to be consulted to accurately simulate usage. However, by changing the problem statement slightly, we demonstrate how prior "saved-state" is incorporated into a usage model. As before the Enter Data option is only available when a project has been selected. The Analyze Data feature is operational when a project is selected *and* data has been entered through the Enter Data option. The Print Report option must have a selected project that has data already entered *and* analyzed before it will function. In order to incorporate this new behavior, we must

model the status of the saved-state. We define the following equivalence classes for the *project defined* state variable: None= no project selected, New= project selected but no data entered, Data= project selected and data entered, Anlz= project selected, data entered, and analysis performed. We then reorganize the STD to account for this new classification. The new diagram appears in Fig. 3.

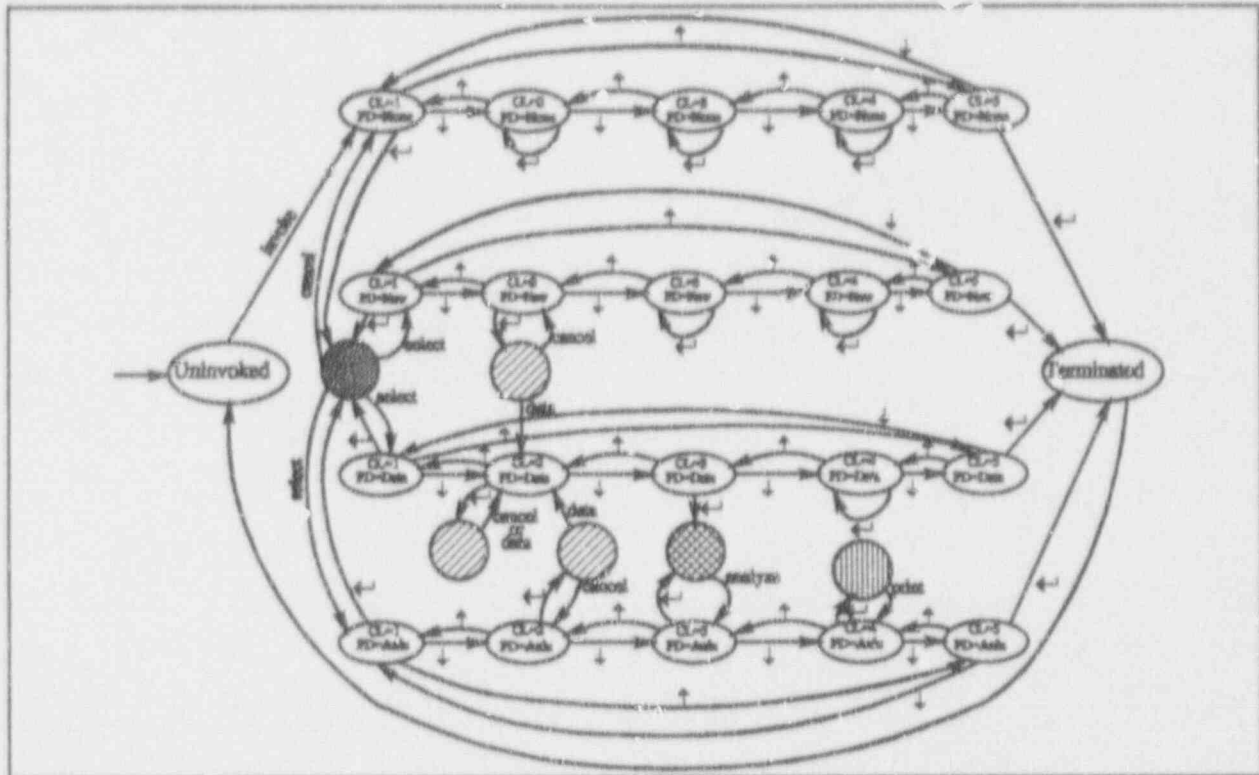


Figure 3: A State Transition Diagram for the Example Selection Menu with Saved-State

The finite-state model of software usage allows complex usage histories to be incorporated into a concise model. As with any finite model of an essentially infinite process, certain assumptions about equivalence classes of events must be made. However, our early experience indicates that the finite-state model performs well with both application and real-time software. Even large software systems have proven amenable to such models.

The *statistics* of the usage model are in the form of probability distributions over the transitions from each state. Information about appropriate probability distributions comes from sources other than the specification, such as captured usage data from prototypes, simulators, or prior software versions. This data is supplemented as necessary via interviews of potential customers to gain intuition into expected usage patterns. The STD model has identified exactly which probabilities must be estimated, i.e., the probability that a given stimulus is applied from a specific usage state. Thus, it is helpful to specify expected usage scenarios and collect statistics accordingly. For example, it may be reasonable to assume that

users of the selection menu will execute the software for three different purposes: (i) to create a new data set, (ii) to run the analysis and print the report for an existing data set, and (iii) to print the report for an existing data set that has previously been analyzed. Scenarios that obey the structure of the model, i.e., sequences of usage states and stimuli, can then be constructed (and even weighted if appropriate) to establish a data set from which relative frequency probability estimates may be obtained. It is important to remember that the initial probability estimates can be refined based on subsequent analysis. The analysis takes the form of probability distributions and properties of random variables that are meaningful to the process of using and testing software. For example, the number of stimuli expected per test case and the number of test cases until complete domain coverage occurs are examples of universally applicable measures. The measures support both the reestimation of model probabilities and aid testers in determining stochastic attributes of the model that affect time and budget of the test. Often, adjusting parameters can actually lessen the amount of testing that must be performed without sacrificing test effectiveness.

In order to generate paths statistically, transition probabilities are established that simulate expected usage. The exit arc probabilities from each state must sum to 1. Sequences are automatically generated from the STD by stepping through state transitions (according to the transition probabilities), from Uninvoked to Terminated, and recording the sequence of stimuli on the path traversed.

The usage model format, i.e., the STD, is familiar to many software professionals and, indeed, is a tool of the trade in object and data-oriented methods. This familiar format is a basis for communication of complex relationships and eases review of the test plan. Additional software functionality, e.g., a new feature or version, is easily incorporated into the model with the addition of states and addition/redirection of arcs.

There are two distinct methods of analysis for two different situations in testing. The first situation is when failures occur. The occurrence of failures means that the software is incorrect and must be reengineered. When failures are discovered, a new Markov process is defined that incorporates all test data from the last engineering change (i.e., code fix) including all failures observed. This *testing chain* is constructed as follows.

Given a usage model U with transition probabilities U_{ij} we define a testing model T . The initial testing model T_0 is a copy of usage model U with all transition probabilities set to 0. When failure-free sequences are executed a sequence of testing models is defined. T_1 is obtained from T_0 by incrementing arc frequencies along the path of states from invocation of the software to termination in test sequence s_1 . Similarly, T_2 is obtained from T_1 by sequence s_2 ; and, in general, T_i is obtained from T_{i-1} by sequence s_i . In this way, frequency counts on arcs in T_i are always obtained from specific test sequences applied to the software. These arc frequencies are converted to relative frequency probabilities whenever computations with T_i 's state transition probabilities are required.

Suppose now that failures do occur and that the j^{th} failure f_j is detected during input of test sequence s_i to the software. To incorporate this failure event into the test history, a new state labeled f_j is placed in Markov chain T_i exactly at its location in s_i . The arcs to/from new state f_j have frequency count 1. If f_j is a catastrophic failure, then the test sequence is aborted and the arc from f_j goes to the terminal state; otherwise, the test sequence can continue and the arc from f_j goes to the next state in s_i . In this way, T_i is maintained as a Markov chain that incorporates both the underlying structure of the source of test sequences, U , and the frequency-count history of sequences-plus-failures as testing evolves. In other words, T models every event that occurs in the testing process and recognizes the dependencies on each event as dictated by the usage model.

What can be said about the series T_0, T_1, \dots, T_m ? If no failures are detected, the evolution of T is dictated solely by test sequences from U . The Strong Law of Large Numbers for Markov chains guarantees (with probability 1) that these sequences s_1, \dots, s_m will become statistically typical of U when enough are generated. This means that convergence of T to U is certain because the relative frequencies on T 's arcs will converge to the probabilities on U 's arcs. A key point is that the test history T is statistically typical of the usage chain U if and only if convergence is achieved.

In other words, U is a fixed reference toward which T_i evolves at an expected rate with statistical variation that depends on factors such as the source entropy of U . This evolution is well-controlled and predictable in statistical terms.

Convergence of T to U is adversely affected by failures of the software during testing. To achieve convergence when failures have been observed, the relative-frequency probabilities on arcs to failure states in T_i must approach 0. In this way, the probabilities on the non-failure arcs are still forced to converge to the corresponding (non-zero) values in U . If even one failure occurs, this can only be accomplished when the software responds to more test sequences without exhibiting failures. Thus, failures automatically impose additional testing to overcome their adverse impact on the convergence of T to U .

Assuming that an infinite number of failures is impossible, convergence will ultimately be achieved. Intuitively, comparison of the actual evolution of T (including failures) with its expected evolution (without failures) supports statistical estimation of the software's characteristics based on the software's actual performance. At any point in the testing process, the most recent test history T_i is available for analysis. Because T_i itself is a well-defined Markov chain, computations are based on the theory of Markov chains.

We have defined a suite of computations on the test history that allow reliability estimation and give testers insight into the representativeness of the sample of test cases.

The testing chain can be interpreted as a square matrix with the states as indices and the transition probabilities as entries. This matrix is called the transition matrix and is denoted P .

with entries p_{ij} . The sum of the entries for each row in this matrix is exactly 1.

The reliability is computed by forcing absorption to occur at the terminate state as well as at each failure state. These changes yield a new transition matrix P' . Any sequence of states beginning at the initial state of P' will be absorbed in either the terminate state or one of the failure states. The reliability, R , is the probability that absorption occurs at terminate and is computed

$$R_{Inv, Term} = \hat{P}_{Inv, Term} + \sum_{j \in \tau} \hat{P}_{Inv, j} R_{j, Term} \quad (1)$$

where Inv denotes invocation of the software, $Term$ denotes the terminate state, and τ is the set of transient (non-absorbing) states. Note that $1/R$ is the expected number of sequences of the testing chain until a failure state appears.

The *MTBF* is a non-standard analytical result that is derived by Whittaker in [13]. It is computed as follows.

$$MTBF = \sum_{i \in f_1, \dots, f_n} v_i \left(\sum_{j \in u_1, \dots, u_n} \hat{P}_{ij} (m_j + 1) \right)$$

where v_i is the renormalized long run probability for failure state f_i , m_j is the mean time until the first occurrence of any failure state from j , u_1, \dots, u_n is the set of usage states, and f_1, \dots, f_n is the set of failure states.

The *discrimination* of the testing chain from the usage chain, denoted $D(U, T)$, is computed by

$$D(U, T) = \sum_{i, j} \pi_i p_{ij} \log \frac{p_{ij}}{\hat{p}_{ij}}$$

where π is the usage distribution, p_{ij} is the probability of a transition from i to j in the usage chain, and \hat{p}_{ij} is the corresponding probability in the testing chain.

The *distance* between U and T_i is

$$d(U, T_i) = \sqrt{\sum_{j, k} (p_{jk} - q_{jk})^2}$$

Both the distance and the discrimination allow the analysis of the convergence of $T \rightarrow U$ and indicate the representativeness of the sample test sequences, i.e., the degree to which they are statistically typical of U .

References for Appendix A

- [1] P. A. Currit, M. Dyer, H. D. Mills, "Certifying the reliability of software," IEEE Trans on Software Engineering, Vol SE-12, No 1, January 1986, pp 3-11
- [2] R. C. Linger, H. D. Mills, "A case study in Cleanroom software engineering: The IBM COBOL structuring facility," Proceedings of COMPSAC'88, IEEE 1988
- [3] R. C. Linger, H. D. Mills, B. I. Witt, Structured Programming: Theory and Practice, Addison Wesley 1979
- [4] H. D. Mills, "The new math of computer programming," Comm of the ACM, Vol 18, No 1, 1975
- [5] H. D. Mills, "Stepwise refinement and verification in box-structured systems," IEEE Computer, June 1988, pp 23-36
- [6] H. D. Mills, M. Dyer, R. C. Linger, "Cleanroom software engineering," IEEE Software, September 1987, pp 19-24
- [7] H. D. Mills, R. C. Linger, A. R. Hevner, Principles of Information Systems Analysis and Design, Academic Press, 1986
- [8] H. D. Mills, R. C. Linger, A. R. Hevner, "Box structured information systems," IBM Systems Journal, Vol 26, No 4, 1987, pp 395-413
- [9] D. L. Parnas, "A technique for software module specification with examples", CACM 15, 5, May 1972, pp. 330-336
- [10] D. L. Parnas, "Designing software for ease of extension and contraction," IEEE Trans on Software Engineering, SE-5, No 3, March 1979, pp 128-138
- [11] D. L. Parnas, Y. Wang, "The Trace assertion method of module-interface specification", Tech Rep. 89-261, Queen's University, TRIO, October 1989
- [12] N. Wirth, "Program development by stepwise refinement", CACM 14, 4, April 1971, pp 221-227
- [13] J. A. Whittaker, Markov Chain Techniques for Software Testing and Reliability Analysis, Ph.D. Dissertation, Department of Computer Science, University of Tennessee, Knoxville, 1992.

Appendix B

The Power of Usage Testing over Coverage Testing

The insights and data of Adams [1] in the analysis of software testing, and the differences between software errors and failures, give entirely new understandings in software testing. Since Adams has discovered an amazingly wide spectrum in failure rates for software errors, it is no longer sensible to treat errors as homogeneous objects to find and fix. Finding and fixing errors with high failure rates produces much more reliable software than finding and fixing random errors, which generally have average or low failure rates.

The major surprise in Adams' data is the relative power of finding and fixing errors in usage testing over coverage testing, a factor of 30 in increasing MTTF. That factor of 30 seems incredible until the facts are worked out from Adams' data. But it explains many anecdotes about experiences in testing. In one such experience, an operating systems development group used coverage testing systematically in a major revision and for weeks found mean time to abends in seconds. It reluctantly allowed user tapes in one weekend, but on fixing those errors, found the mean time to abends jumped literally from seconds to minutes.

The Adams data is given in Table 1 from [1]. It describes distributions of failure rates for errors in 9 major IBM products, including the major operating systems, language compilers, data base systems. The uniformity of the failure rate distributions among these very different products is truly amazing. But even more amazing is a spread in failure rates over 3 orders of magnitude, from 19 months to 5000 years (60 K months) calendar time in MTTF, with about a third of the errors having an MTTF of 5000 years, and 1% having an MTTF of 19 months.

TABLE 1
DISTRIBUTIONS OF ERRORS (IN %) AMONG
MEAN TIME TO FAILURE (MTTF) CLASSES

	MTTF in K months							
	60	19	6	1.9	.6	.19	.06	.019
Product								
1	34.2	28.8	17.8	10.3	5.0	2.1	1.2	.7
2	34.2	28.0	18.2	9.7	4.5	3.2	1.5	.7
3	33.7	28.5	18.0	8.7	6.5	2.8	1.4	.4
4	34.2	28.5	18.7	11.9	4.4	2.0	.3	.1
5	34.2	28.5	18.4	9.4	4.4	2.9	1.4	.7
6	32.0	28.2	20.1	11.5	5.0	2.1	.8	.3
7	34.0	28.5	18.5	9.9	4.5	2.7	1.4	.6
8	31.9	27.1	18.4	11.1	6.5	2.7	1.4	1.1
9	31.2	27.6	20.4	12.8	5.6	1.9	.5	.0

With such a range in failure rates, it is easy to see that coverage testing will find the very low failure rate errors a third of the time with practically no effect on the MTTF by the fix, whereas usage testing will find many more of the high failure rate errors with much greater effect. Table 2 develops the data, using Table 1, that shows the relative effectiveness of fixes in usage testing and coverage testing, in terms of increased MTTF. Table 2 develops the change in failure rates for each MTTF class of Table 1, because it is the failure rates of the MTTF classes that add up to the failure rate of the product.

TABLE 2
ERROR DENSITIES AND FAILURE DENSITIES
IN THE MTTF CLASSES OF TABLE 1

Property								
M	60	19	6	1.9	.6	.19	.06	.019
ED	33.2	28.2	18.7	10.6	5.2	2.5	1.1	.5
ED/M	.6	1.5	3.1	5.6	8.7	13.2	18.3	26.3
FD	.8	2.0	3.9	7.3	11.1	17.1	23.6	34.2
FD/M	0	0	1	4	18	90	393	1800

First, in Table 2, line 1, denoted M (MTTF), is repeated directly from Table 1, namely the mean time between failures of the MTTF class. Next, line 2, denoted ED (Error Density), is the average of the error densities of the 9 products of Table 1, column by column, which represents a typical software product. Line 3, denoted ED/M, is the contribution of each class, on the average, in reducing the failure rate by fixing the next error found by coverage testing (1/M is the failure rate of the class, ED the probability a member of this class will be found next in coverage testing, so their product, ED/M, is the expected reduction in the total failure rate from that class). Now ED/M is also proportional to the usage failure rate in each class, since failures of that rate will be distributed by just that amount. Therefore, this line 3 is normalized to add to 100% in line 4, denoted FD (Failure Density). It is interesting to note that Error Density (ED) and Failure Density (FD) are almost reverse distributions, Error Density about a third at the high end of MTTFs and Failure Density about a third at the low end of MTTFs. Finally, line 5, denoted FD/M, is the contribution of each class, on the average, in reducing the failure rate by fixing the next error found by usage testing.

The sums of the two lines ED/M and FD/M turn out to be proportional to the decrease in failure rate from the respective fixes of errors found by coverage testing and usage testing, respectively. Their sums are 77.3 and 2306, with a ratio of about 30 between them. That is the basis for the statement of their relative worth in increasing MTTF. It seems incredible at first glance, but that is the number!

To see that in more detail, consider, first, the relative decreases in failure rate R in the two cases:

Fix next error from coverage testing

$$R \rightarrow R - (\text{sum of ED/M values})/(\text{errors remaining}) \\ \approx R - 77.3/E$$

Fix next error from usage testing

$$R \rightarrow R - (\text{sum of FD/M values})/(\text{errors remaining}) \\ = R - 2306/E$$

Next, the increase in MTTF in each case will be

$$1/(R - 77.3/E) - 1/R = 77.3/(R*(E*R - 77.3))$$

and

$$1/(R - 2306/E) - 1/R = 2306/(R*(E*R - 2306))$$

In these expressions, the numerator values 77.3 and 2306 dominate, and the denominators are nearly equal when $E*R$ is much larger than 77.3 or 2306 (either $77.3/(E*R)$ or $2306/(E*R)$ is the fraction of R reduced by the next fix and is supposed to be small in this analysis). As noted above, the ratio of these numerators is about 30 to 1, in favor of the fix with usage testing.

References for Appendix B

- [1] E. N. Adams, "Optimizing preventive service of software products," IBM Journal of Research and Development, January 1984.

Appendix C

Zero Defect Software is Really Possible

In spite of the experiences of this first human generation in software development, zero defect software is really possible. However, there is no foolproof logical way to know that software is zero defect. The proof is in the using of the product without ever finding any failures. Mathematics is very helpful in creating software that executes with zero defects. But it is insufficient to guarantee it, in part due to human fallibility in using mathematics, in part to mathematics itself in logical incompleteness. Statistics is also very helpful in creating software that executes zero defect, but is also insufficient. For example, given a program that has been tested statistically without failures, say for time T of execution, a minimax argument permits a statement that estimates the Mean Time to Failure (MTTF) as $2T$. But the program then may well be used many times T with no failures, and the MTTF goes up accordingly. The proof is in the usage.

Three illustrations of zero defect software are discussed in the following paragraphs. First, the US 1980 Census was acquired by a nationwide network system of 20 miniprocessors. The system was controlled by a 25 KLOC program, which operated its entire ten months in field use with no failure observed. It was developed by Paul Friday, of the US Census Bureau, using stepwise refinement and functional verification in Pascal. Mr. Friday was given the highest technical award of the US Department of Commerce for that achievement.

Second, the IBM Wheelwriter typewriter products released in 1984 are controlled by three microprocessors with a 65 KLOC program. It has had millions of users ever since with no failures detected. The IBM team creating this software also used functional verification and extensive testing in a well managed environment to achieve this result.

Third, the US space shuttle software of some 500 KLOC, while not completely zero defect, has been zero defect in all flights. The IBM team also used functional stepwise refinement and verification, and extensive testing to achieve that result. The space shuttle software is such a large, complex, and visible product that there are real lessons in it. As noted, all programmers were required to complete a basic curriculum of six pass/fail courses in understanding programs as rules for mathematical functions, and functional verification of programs and modules [4].

Cleanroom Engineering of Software has been demonstrated as feasible and beneficial in both productivity and quality in software development. The idea driving the development of Cleanroom Engineering was that by first developing and then applying proper mathematical rigor to the engineering of software, essentially failure-free software could be developed. The effort started some 25 years ago at IBM. Today, leading edge organizations are using one or more of the Cleanroom technologies to develop software. The engineers utilizing Cleanroom Engineering are developing significant software systems that are of very high quality. There are several examples of widely used failure-free software. Since software development is a

human activity subject to human fallibilities, even Cleanroom software may contain latent failures.

Current quality metrics in terms of defect rate per 1000 Lines Of Code (KLOC) at delivery for software development organizations are as follows: for organizations who are using traditional practices (bottom-up design, structured code, defect removal through testing) 15-18 defects/KLOC; modern practices (top-down design, structured programming, design/code inspections, et.) 2-4 defects/KLOC; and Cleanroom Engineering (0-1 defects/KLOC). Users of Cleanroom are now targeting to release 100 KLOC systems that, more often than not, have no failures ever found during execution of the software. They expect, as they move up the learning curve, to do even better.

The surprise to many is that Cleanroom developers also increase their productivity at the same time that they are achieving these much higher quality levels. Productivity rates of 750 lines of code per staff month are common with Cleanroom projects compared to the typical 70 - 250 lines of code per staff month being achieved by developers using modern programming practices. The reason is that it is less costly to design the software in a manner that will prevent failures than it is to test in quality into a failure laden product, where defects have been designed in as a result of using inadequate engineering practices. Cleanroom developers expect to soon be achieving a productivity of 1500 lines of code per staff month as they progress up the learning curve.

Early Cleanroom Experiences

Some of the early Cleanroom projects that were conducted to confirm the ideas are summarized in this section.

The IBM COBOL Structuring Facility, a complex product of some 80K lines of PL/I source code, was developed in the Cleanroom discipline, with box structured design and functional verification but no debugging before usage testing and certification of its correctness. A version of the USAF HH60 (helicopter) flight control program of over 30 KLOC was also developed using Cleanroom. The Coarse/Fine Attitude Determination Subsystems (CFADS) of the UARS Attitude Ground Support System (AGSS) of some 30 KLOC has been developed with Cleanroom at NASA.

The IBM COBOL Structuring Facility (SF) converts an unstructured COBOL program into a structured one of identical function. It uses considerable artificial intelligence to transform a flat structured program into one with a deeper hierarchy that is much easier to understand and modify. The product line was prototyped with Cleanroom discipline at the outset, then individual products were generated in Cleanroom extensions. In this development, several challenging schedules were defined for competitive reasons, but every schedule was met.

The COBOL/SF products have high function per line of code. The prototype was estimated at 100 KLOC by an experienced language processing group, but the Cleanroom developed prototype was 20 KLOC. The software was designed not only in structured programming, but also in structured data access. No arrays or pointers were used in the design; instead, sets, queues, and stacks were used as primitive data structures [5]. Such data structured programs are more reliably verified and inspected, and also more readily optimized with respect to size or performance, as required.

COBOL S/F, Version 2, consisted of 80 KLOC, 28 KLOC reused from previous products, 52 KLOC new or changed, designed and tested in a pipeline of five increments [3], the largest over 19 KLOC. A total of 179 corrections were required during certification, under 3.5 corrections per KLOC for code with no developer execution. The productivity of the development was 740

LO per staff month, including all specification, design, implementation, and management, in meeting a very short deadline.

The HH60 flight control program was developed on schedule. Programmers' morale went from quite low at the outset ("why us?") to very high on discovering their unexpected capability in accurate software design without debugging. The twelve programmers involved had all passed the pass/fail course work in mathematical (functional) verification of the IBM Software Engineering Institute, but were provided a week's review as a team for the project. The testers had much more to learn about certification: by objective statistics [1].

The subsystem Coarse/Fine Attitude Determination System (CFADS) of the NASA Attitude Ground Support System (AGSS) of some 30 KLOC was developed in Fortran. 62% of the subroutines, which averaged 258 source lines each, compiled correctly the first time the testers tried to compile it, with but one of the rest compiled correctly on the second attempt. Compared with well measured related systems, the failure rate was down by a factor of 5 while the productivity was up by 70% [2].

V. R. Basili and F. T. Baker introduced Cleanroom ideas in an undergraduate software engineering course at the University of Maryland, assisted by R. W. Selby. As a result, a controlled experiment in a small software project was carried out over two academic years, using fifteen teams with both traditional and Cleanroom methods. The result, even on first exposure to Cleanroom, was positive in the production of reliable software, compared with traditional results [7].

Cleanroom projects have been carried out at the University of Tennessee, under the leadership of J. H. Poore [6] and at the University of Florida under H. D. Mills. At Florida, seven teams of undergraduates produced uniformly successful systems for a common structured specification of three increments. It is a surprise for undergraduates to consider software development as a serious engineering activity using mathematical verification instead of debugging, since software development is typically introduced primarily as a trial and error

activity with no real technical standards.

References for Appendix C

- [1] P. A. Currit, M. Dyer, H. D. Mills, "Certifying the reliability of software," IEEE Trans on Software Engineering, Vol SE-12, No 1, January 1986, pp 3-11
- [2] A. Kouchakdjian, Scott E. Green, V. R. Basili, "The Cleanroom Case Study in the Software Engineering Laboratory: An Experiment in Formal Methods", SEL, University of Maryland 1989
- [3] R. C. Linger, H. D. Mills, "A case study in Cleanroom software engineering: The IBM COBOL structuring facility," Proceedings of COMPSAC'88, IEEE 1988
- [4] R. C. Linger, H. D. Mills, B. I. Witt, Structured Programming: Theory and Practice, Addison Wesley 1979
- [5] H. D. Mills, R. C. Linger, "Data structured programming: Program design without arrays and pointers," IEEE Trans on Software Engineering, Vol SE-12, No 2, February 1986, pp 192-197
- [6] H. D. Mills, J. H. Poore, "Bringing software under statistical quality control," Quality Progress, November 1988, pp 52-55
- [7] R. W. Selby, V. R. Basili, F. T. Baker, "Cleanroom software development: An empirical evaluation," IEEE Trans on Software Engineering, Vol SE-13, No 9, September 1987

Appendix D
Notes on Leveson's talk August 21, 1992 to NRC, pp 412-459

For ACRS Meeting
Washington, D. C.
February 9, 1993

Harlan D. Mills
Software Engineering Technology, Inc.

Nancy Leveson, professor at U. California, Irvine, moving to U. Washington as endowed professor, has unique experience and capability in evaluating software quality. She has a broad view of software, from an engineering view rather than simply computer science.

My assessment is that she approaches software quality intuitively, does not believe formal methods are practical, and has invented some informal ideas in both software and management to take their place. She does not like the term "correctness", but is really concerned with "software correctness" in both the "explicit software" written to meet a specification and the "implicit software" not written but also needed to meet the real need. The terms "explicit" and "implicit" are mine here. She has many useful ideas, but I think she also has some critical ideas that keep verification and correctness intuitive and possibly faulty. Humans can make mistakes when using complete methods. For example mistakes are possible in doing arithmetic in arabic digits, but humans will make even more mistakes in roman numerals.

I illustrate these differences in some key points, which are pulled out of a broader report that follows as an Appendix. These points each begin with a quotation of Professor Leveson, followed by a response of mine.

4. "I don't think software engineers have been very aware that there are conflicts; they just think you can sort of make perfect software, you know, and "correct" in quotes, and there's no such thing as correct software anymore than there's any such thing as correct system or correct nuclear power plants." p 414, ls 13-18

I'd modify point 4 as follows.

- 4'. *Software engineers need additional education and rigor to create perfect software which is correct to given specifications, which are complete for what is needed. to replace p 414, ls 13-18*

Next, Professor Leveson makes a point

6. "In computer involved accidents, ..., almost all of them are due to inadequate

design foresight in requirements specifications." p 415, ls 6-9

I disagree and see many accidents due to previous fixes of other failures.

6'. *In computer involved accidents, many of them are due to inadequate design foresight in requirements specifications, but even more are due to previous fixes. to replace p 415, ls 6-9*

I believe software design and software testing should be done by different groups, with strong software proofs carried out by developers before passing it over for certification. There should be but a few, if any, failures discovered, but failures should be returned for correction to developers who then return the software to the certifiers for retesting.

Professor Leveson makes a distinction between safety and failures, as follows.

7. "You know, I think it's a real mistake to, first of all, define safety in terms of failures." p 416, ls 15-16

I think I agree with this, assuming she means safety refers to both referenced requirements and unreferenced requirements. I'd make that more specific as follows.

7'. *Safety references two kinds of correctness, first with respect to the specification, and second to unspecified requirements not in the specification. to replace p 416, ls 15-16*

For example, I'd replace the following sentence with another following.

8. "So I don't even know what correct means, and I try not to use it, but we still do, because I don't know what correct software is." p 418, ls 4-6

8'. *Software systems can be developed to meet specifications and never create an explicit failure. Specifications must also be complete in order to prevent implicit failures. to replace p 418, ls 4-6*

On (p 443, ls 1-6), Professor Leveson returns to concerns with the term correctness.

13. "-- when software engineers say correctness, all we really mean is consistency with the specification, which is not at all correctness, it's only consistency with the specification -- versus verification of safety." p 443, ls 1-6

As noted above, failures, if any, may be explicit from incorrect execution, or implicit from faulty specifications. She then discusses unsafe states that can arise from faulty execution or missing specification parts.

As already stated, I believe correctness should be used in a positive way by including specification completeness for defining safety.

13'. -- when software engineers say correctness, we mean the software is correct with respect to the specification, and that no hidden specifications exist -- then correctness is consistent with safety. to replace p 443, ls 1-6

In closing, Professor Leveson goes back to the fundamentals from her perspective.

20. "most of engineering is learned through trial and error. You have accidents and you learn, and you establish codes and standards of practice." p 458, ls 20-23

Point 20 is debatable. I'd say it as follows.

20'. "most of engineering is intended to avoid trial and error. If you have accidents you learn as you establish codes and standards of practice." to replace p 458, ls 20-23

In summary, Professor Leveson thinks very hard about software correctness, both explicitly and implicitly. I think she is overly intuitive on design and verification, and assumes testing by the designers is an important way to find and remove failures. I think such joint design and testing creates deeper failures no matter how hard people try. Informally created code has a fifteen percent failure rate in corrections because of the lack of formal documentation for the code. As a result, the software seldom becomes correct from informal debugging.

Finally, although not discussed by Professor Leveson, there is a new way to examine the correctness of software by testing and use. If software has many unknown sources of failure, finding one such source and fixing it will reduce (hopefully) the failures possible. So classical calculations of statistics may be useful in estimating possible failures from a fix. But if software is of high quality with very few sources, possible only the one just found, the classical estimate is wrong. In this case a hypothesis that no sources of failure remain can be formulated. If an additional failure is found, the hypothesis has failed. By correcting this new source of failure, a new hypothesis of perfect software can be made. In fact, the classical calculations can be carried out, as well, so nothing is really lost by carrying out both hypotheses.

Clearly, Professor Leveson is legitimately concerned with the quality of estimates of correctness, both explicit and implicit. But I believe the needs of NRC is to make use of statistics and rigorous software design and verification before testing so that effective human engineering is brought to bear.

Appendix

Notes on Leveson's talk August 21, 1992 to NRC, pp 412-459

Her first summary on pp 413-414 is as follows.

1. "... we've got to stop believing hype and overselling about software engineering techniques." p 413, ls 22-24
2. "We need to add safety techniques to software design.", p 414, ls 2-3
3. "We've got to get the software engineers working with the system safety engineers." p 414, ls 8-9
4. "I don't think software engineers have been very aware that there are conflicts; they just think you can sort of make perfect software, you know, and "correct" in quotes, and there's no such thing as correct software anymore than there's any such thing as correct system or correct nuclear power plants." p 414, ls 13-18
5. "From a systems safety engineering standpoint, we've got -- they've got to start including software and stop ignoring it." p 414, ls 19-21

I disagree with point 1. Solid software engineering techniques are needed for effective specification and program design. I'd modify point 1 as follows.

"I' ... we've got to understand and make use of software engineering techniques as they are rigorously known. to replace p 413, ls 22-24

I certainly would not make use of intuitive hopes as often done today, only real engineering techniques.

I disagree with point 4. Software is either correct or not with respect to a specification because it is logical. Systems and nuclear power plants are physical and can only operate correctly with given data at a given moment in time. I'd modify point 4 as follows.

4'. Software engineers need additional education and rigor to create perfect software which is correct to given specifications, which are complete for what is needed. to replace p 414, ls 13-18

Next, Professor Leveson makes a point

6. "In computer involved accidents, ..., almost all of them are due to inadequate design foresight in requirements specifications." p 415, ls 6-9

I disagree and see many accidents due to previous fixes of other failures.

6'. *In computer involved accidents, many of them are due to inadequate design foresight in requirements specifications, but even more are due to previous fixes. to replace p 415, ls 6-9*

I believe software design and software testing should be done by different groups, with strong software proofs carried out by developers before passing it over for certification. There should be but a few, if any, failures discovered, but failures should be returned for correction to developers who then return the software to the certifiers for retesting.

Professor Leveson makes a distinction between safety and failures, as follows.

7. "You know, I think it's a real mistake to, first of all, define safety in terms of failures." p 416, ls 15-16

I think I agree with this, assuming she means safety refers to both referenced requirements and unreferenced requirements. I'd make that more specific as follows.

7'. *Safety references two kinds of correctness, first with respect to the specification, and second to unspecified requirements not in the specification. to replace p 416, ls 15-16*

Professor Leveson refers to correctness, especially Shuttle software, and claims no sizable software systems ever operated zero defect. Shuttle software as operated has been changed for many years to meet new mission requirements. While high quality, it is not zero defect, but it has not created any system failures. But smaller systems have operated completely zero defect. Two well known examples are the 1980 US Census program, of 25,000 lines, and the IBM typewriter program of 63,000 lines. The US Census program ran 10 months to complete the census with no failures at all, small or large. The IBM typewriter program has operated nine years with millions of users with no failures. In both cases, sound software design methods were applied. For example, I'd replace the following sentence with another following.

8. "So I don't even know what correct means, and I try not to use it, but we still do, because I don't know what correct software is." p 418, ls 4-6

8'. *Software systems can be developed to meet specifications and never create an explicit failure. Specifications must also be complete in order to prevent implicit failures. to replace p 418, ls 4-6*

Next, Professor Leveson introduces four areas of concern.

9. "One is in systems requirements specification analysis, we have modelling languages we've been looking at for modelling systems." p 420, ls 5-7

10. "Then we've been looking at system hazard analysis when we model the system, where we determine whether it can reach a hazardous state" p 420, ls 13-15

11. "We are looking at design for safety -- in other words, to design to protect against safety failures" p 420, ls 19-20

12. "And then the last area we have looked at is code verification." p 421, ls 1-2

She points out that these areas were invented in reverse order. She then discusses how much of this kind of analysis can survive commercial standards of productivity.

In this case, the Darlington shutdown software (p 426, ls 9-15) illustrated her concern on how narrowly certification could or should be. Clearly she was concerned that Darlington did not take her complete advice.

Next, she begins a discussion (p 427, ls 4-19) on modeling the black box behavior of the software, its interfaces with the rest of the system, basic assumptions about the behavior of the other components, including failure behavior of the other components. An example is given with a Hughes torpedo project beginning 1980. Another is in air traffic control.

Next, her study of N-Version Programming, in part with John Knight, illustrates the difficulty of finding failure independence with different programs. That seemed a promising idea, but has not worked out as well as people had hoped (p 435-442).

On (p 443, ls 1-6), Professor Leveson returns to concerns with the term correctness.

13. "-- when software engineers say correctness, all we really mean is consistency with the specification, which is not at all correctness, it's only consistency with the specification -- versus verification of safety." p 443, ls 1-6

As noted above, failures, if any, may be explicit from incorrect execution, or implicit from faulty specifications. She then discusses unsafe states that can arise from faulty execution or missing specification parts.

As already stated, I believe correctness should be used in a positive way by including specification completeness for defining safety.

13'. -- when software engineers say correctness, we mean the software is correct with respect to the specification, and that no hidden specifications exist -- then correctness is consistent with safety. to replace p 443, ls 1-6

Next, Professor Leveson goes into fault trees, and more experiences with Darlington. She compared (p 446, ls 25..) fault trees at Darlington with formal verification techniques, from "two

man months" to "30 man years". I find concern on whether the measurements are comparable. She states the following.

14. "So it was cheap and its also a good protection because it doesn't start from the requirements; it starts from the system fault tree, from the hazard." p 447, ls 3-5

Darlington "made about 42 changes to the software as a result of the fault tree analysis." p 448, ls 13-14 Another example on a UC Berkeley satellite found more faults as follows.

15. "I can't believe it. I found the fault, an error in the software that would have caused the software to destroy the spacecraft." p 449, ls 19-21

Professor Leveson states that in retrospect, they had been looking at the software in a different way that hid the failure. I'd be concerned that the methods were formal enough. But that should be examined.

On p 451, ls 17-22 another comment on Apollo on a accident that might have occurred. It was doubted astronauts would have taken a simulation seriously to reveal a failure.

Next, some discussion about regulatory and policy issues.

In closing, Professor Leveson goes back to the fundamentals from her perspective.

16. "Encourage interaction and cooperation between the software developers and the system safety personnel." p 455, ls 10-12
17. "we have got to be very careful about getting rid of hardware interlocks ... and replacing them with software." p 456, ls 3-6
18. "this cost them (Darlington) so many millions of dollars to get this software certified, and they had to go through this horrible verification technique, which wasn't proposed by me, it was proposed by David Parnas, cost them incredible amounts of money." p 457, ls 8-13
19. "in terms of personnel requirements, we've got to train people better." p 458, ls 1-2
20. "most of engineering is learned through trial and error. You have accidents and you learn, and you establish codes and standards of practice." p 458, ls 20-23

In point 18, there is clearly a debate between Professor Leveson and Professor Parnas about levels of specification and correctness which is well taken.

In point 19 I believe personnel must be educated better as real software engineers who know how to create specifications rigorously, and design and verify software to meet them.

Point 20 is debatable. I'd say it as follows.

20'. "most of engineering is intended to avoid trial and error. If you have accidents you learn as you establish codes and standards of practice." to replace p 458, ls 20-23

In summary, Professor Leveson thinks very hard about software correctness, both explicitly and implicitly. I think she is overly intuitive on design and verification, and assumes testing by the designers is an important way to find and remove failures. I think such joint design and testing creates deeper failures no matter how hard people try. Informally created code has a fifteen percent failure rate in corrections because of the lack of formal documentation for the code. As a result, the software seldom becomes correct from informal debugging.

Analog-To-Digital I&C Conversion
Activities At Connecticut Yankee

Advisory Committee On Reactor Safeguards

February 9, 1993

Northeast Utilities

Attendees

- | | |
|-----------------------|-------------------------------|
| • M. P. Bain | Manager, CY Engineering |
| • M. H. Brothers | Supervisor, CY Engineering |
| • T. G. Cleary | Engineer, Nuclear Licensing |
| • J. G. Fougere | Engineer, CY PSD |
| • J. E. Leger | Engineer, CY I&C |
| • G. R. Pitman | Manager, Project Services |
| • G. P. vanNoordennen | Supervisor, Nuclear Licensing |

Agenda

- Introduction
- Need for Upgrades
- 50.59 Process Used for Upgrades
- Comments on Draft Generic Letter
- Summary

Introduction

- CY has Completed 25 Years of Commercial Operation
- Twice Set World Record for Continuous Operation
- First Light Water Reactor to Achieve 2 Runs over 400 Days
- Presently Over 325 Days of Continuous Operation
- Excellent SALP Ratings

Introduction

- First Analog-To-Digital Upgrade Completed Over 5 Years Ago
- Reactor Protection System Phase I and II Completed in 1987 and 1989
- Auxiliary Feedwater Upgrade Completed in 1991
- Main Feedwater (Reactor Protection System Phase III) Scheduled for May 1993
- Excellent Reliability Experience
- Commercial Dedication Successful

Reactor Protection System

- Three Phase Approach
 - Phase I "Front End" Work - *Sensor*
 - Phase II Logic Replacement - *logic*
 - Phase III Feedwater Control System

Auxiliary Feedwater

- Done to Eliminate Dependence on Control Air
- Utilized Woodward Model 501 Digital Control System
- Was Determined to be a USQ Due to Steam Admission Valve Failure Mode

Reasons for Upgrades

- Obsolete Equipment
- Appendix "R"
- NIS
- ISAP

Method of Upgrade

- 50.59 Process - Not a USQ
- Failure Modes and Effects Analysis Supported 50.59 Determination
- SER Issued for Phase I and II with "caveats"

Feedwater Upgrade

- Safety Related and Control System Upgrade
- Added Required Channels for Single Failure Criterion
- Same Philosophy as Phase I and II (Spec 200M)
- Rosemount Transmitters
- Prior Approval of Tech Spec but not a USQ

Feedwater Control

- Primary and Secondary Systems for Fault Tolerance
- Median Select for Level Input
- Deviation Alarms

Comments on Draft Generic Letter

- NU Supported NUMARC Comments
- Common Mode Software Failure is a Design Issue
- Need to Focus on System Response to Determine FMEA
- Software Failure not a Credible Event in Simple Systems that Have Many Years of Industrial Operational Experience.
- A Defense-In-Depth Analysis is Recommended for RPS and ESFAS I&C Conversions

Comments on Draft Generic Letter

- Information on EMI/RFI Levels at the Installed Location is Needed
- Experience has Found Fields Within Design Parameters
- Reasonable Standards are Needed
- Administrative Controls are Acceptable
- Site Specific Surveys Should Not be Required for all Applications

Summary

- Experience with Analog-To-Digital Conversion has been Excellent
- Encourage Continued Staff and NUMARC Dialogue
- Reasonable Standards and Regulatory Guidance are Needed
- Let the Utility Decide when the Conversion is a USQ

DIGITAL UPGRADE
OF
RADIATION MONITORING SYSTEM
ANALOG SUB-COMPONENTS

R. D. PLAPPERT
Engineering Supervisor
Controls Discipline



Southern California Edison Company
San Onofre Nuclear Generating Station
Units 1, 2 & 3

DIGITAL UPGRADE
OF
RADIATION MONITORING SYSTEM
ANALOG SUB-COMPONENTS

- I. BACKGROUND
- II. ANALOG TO DIGITAL UPGRADE
- III. REASON FOR UPGRADE
- IV. ENGINEERING, QUALIFICATION AND TESTING
- V. IMPLEMENTATION
- VI. LICENSING DOCUMENT CHANGES
- VII. POTENTIAL ANALOG TO DIGITAL UPGRADES

BACKGROUND

- AFFECTS APPROXIMATELY 45 CHANNELS OF
PROCESS AND EFFLUENT MONITORS

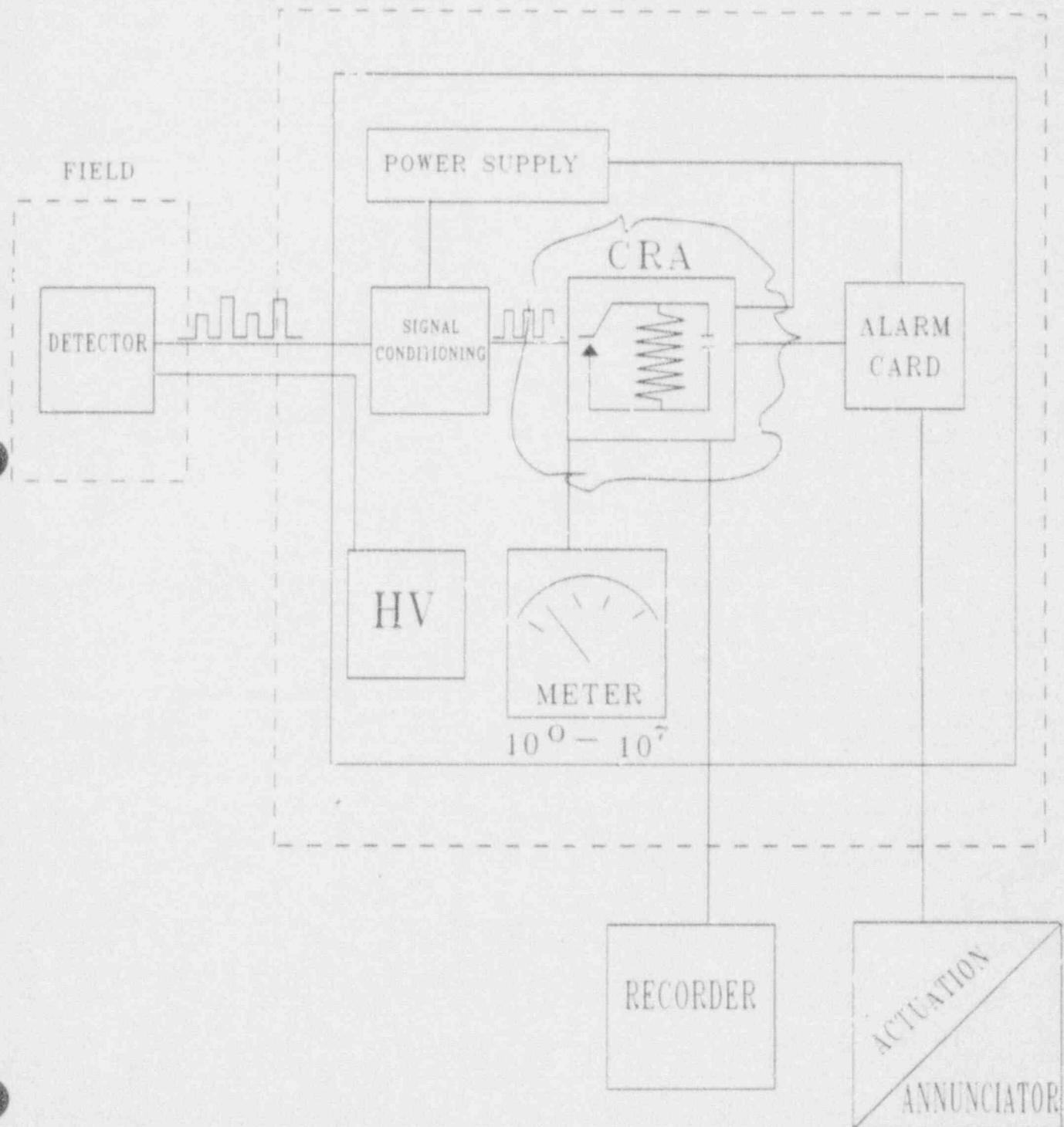
- SAFETY RELATED MONITORING SYSTEMS
 - CONTAINMENT VENTILATION ISOLATION
 - FUEL HANDLING BUILDING VENTILATION
ISOLATION
 - CONTROL ROOM VENTILATION ISOLATION

- RADIOACTIVE EFFLUENT MONITORING SYSTEMS
 - LIQUID EFFLUENT MONITORS
 - GASEOUS EFFLUENT MONITORS

BACKGROUND (Cont.)

TYPICAL RMS BLOCK DIAGRAM

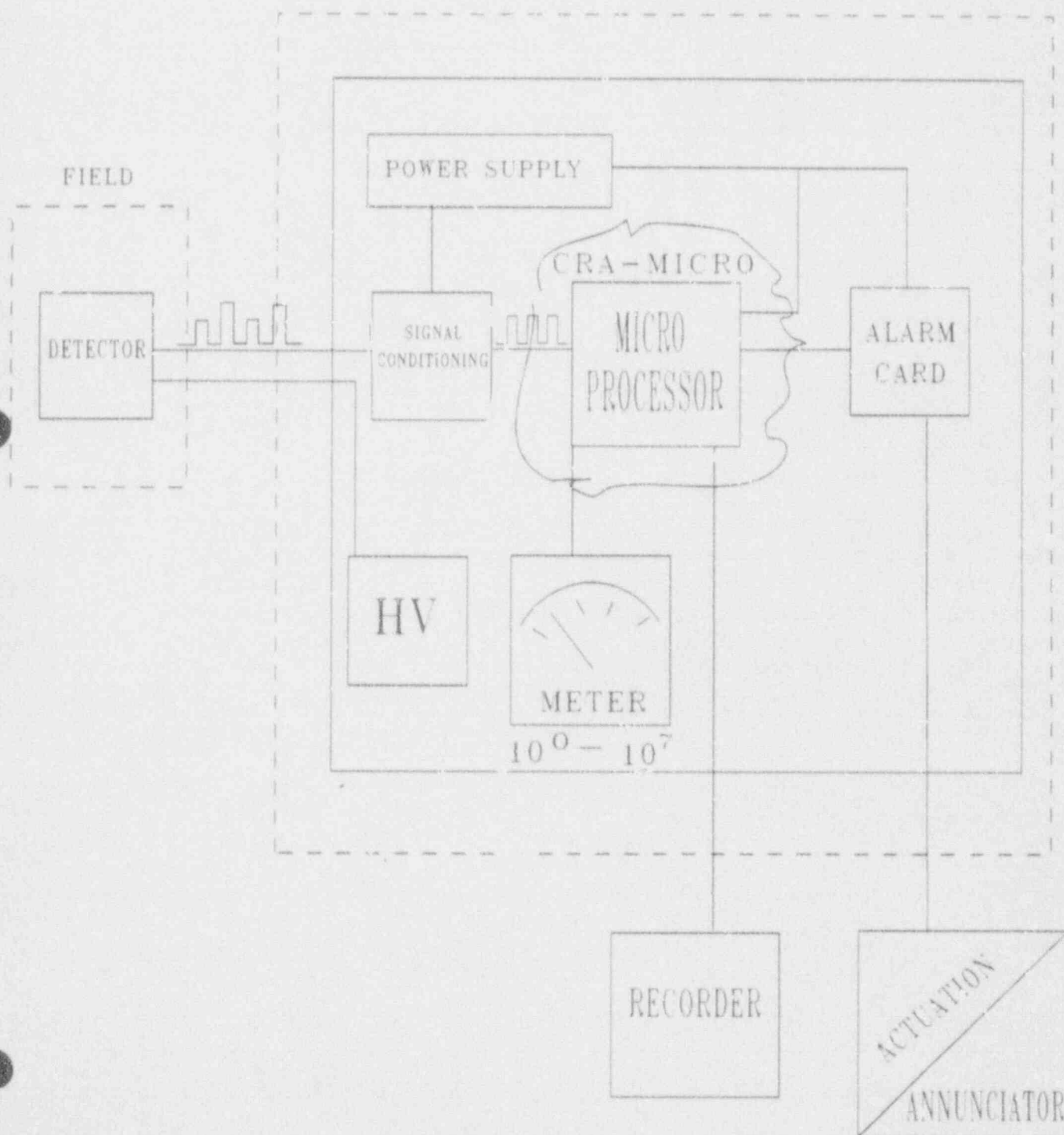
CONTROL ROOM MODULE



ANALOG TO DIGITAL UPGRADE

FUTURE RMS BLOCK DIAGRAM

CONTROL ROOM MODULE



REASON FOR UPGRADE

- OBSOLETE DESIGN
- UNAVAILABILITY OF SPARE PARTS
- CALIBRATION DIFFICULTIES
- HIGH MAINTENANCE COST
- LOW SYSTEM AVAILABILITY
- SPURIOUS SAFETY SYSTEM ACTUATIONS
- SUBSTANTIAL IMPROVEMENT IN ALL AREAS FOLLOWING UPGRADE

ENGINEERING AND EQUIPMENT QUALIFICATION

HARDWARE REQUIREMENTS

CODES AND STANDARDS

- IEEE 323-1974 AND REG GUIDE 1.89 - CLASS 1E ELECTRICAL EQUIPMENT
- IEEE 344-1975 AND REG GUIDE 1.100 - SEISMIC QUALIFICATION OF ELECTRIC EQUIPMENT

SOFTWARE REQUIREMENTS

- SOFTWARE REQUIREMENTS SPECIFICATION (IEEE 830)
- SOFTWARE DESIGN DESCRIPTION (IEEE 1016)
- SOFTWARE VERIFICATION AND VALIDATION (IEEE 1012)

TEST PLAN

- BURN-IN
- FUNCTIONAL TEST
- AGING TO END OF LIFE (MECHANICAL AND RADIATION)
- ENVIRONMENTAL (TEMPERATURE AND HUMIDITY)
- SEISMIC

FAILURE MODES AND EFFECTS ANALYSIS (FMEA) (IEEE 352)

IMPLEMENTATION

- QUALIFICATION TEST REPORT BEING PREPARED
- SOFTWARE V & V IN FINAL STAGES OF COMPLETION
- ADDITIONAL TESTING (ISOTOPIC)
- UPGRADES TO BE IMPLEMENTED DURING ROUTINE SURVEILLANCE
- APPROVE DESIGN CHANGE IN APRIL 1993

LICENSING DOCUMENT CHANGES

- UFSAR CHANGE
- TECHNICAL SPECIFICATION - UNDER REVIEW

POTENTIAL ANALOG TO DIGITAL UPGRADES

- COMPLETE RADIATION MONITORING
SYSTEM UPGRADE
- TOXIC GAS ISOLATION SYSTEM UPGRADE

5

Implementation of
Eagle 21 Protection
System Upgrade at
Sequoyah 1



Ron Gladney
Tennessee Valley Authority

Reasons to Implement Digital Upgrade

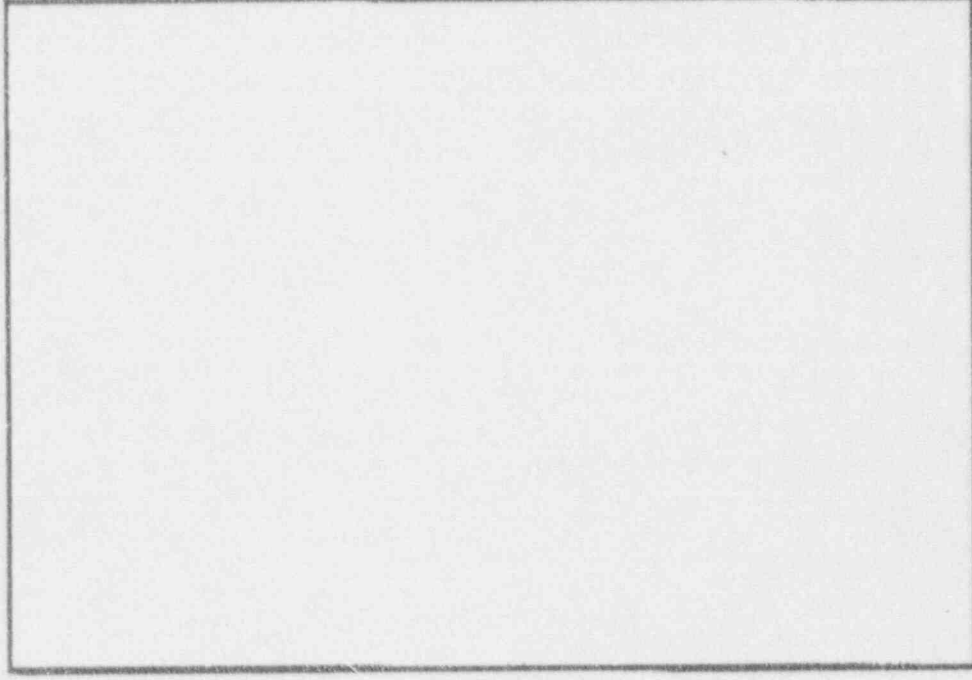
- Implementation of RTD bypass manifold elimination
- Aging analog hardware
- Significant manpower needed to calibrate and maintain analog system
- Analog system calibration drift (reactor coolant narrow range temperature)
- Improved accuracy with digital
- Protection system logic upgrade

Reasons to Implement Digital Upgrade

- Implementation of trip reduction measures
- Capability for automated surveillance testing
- Capability to bypass a channel for testing
- Simplified implementation of Reg Guide 1.97
- Flexibility for transmitter upgrades from 10-50 MA to 4-20 MA

Considerations in Performing a Digital Upgrade to the Plant

- Design
- Maintenance
- Procedures
- Operations
- Training



Design Considerations

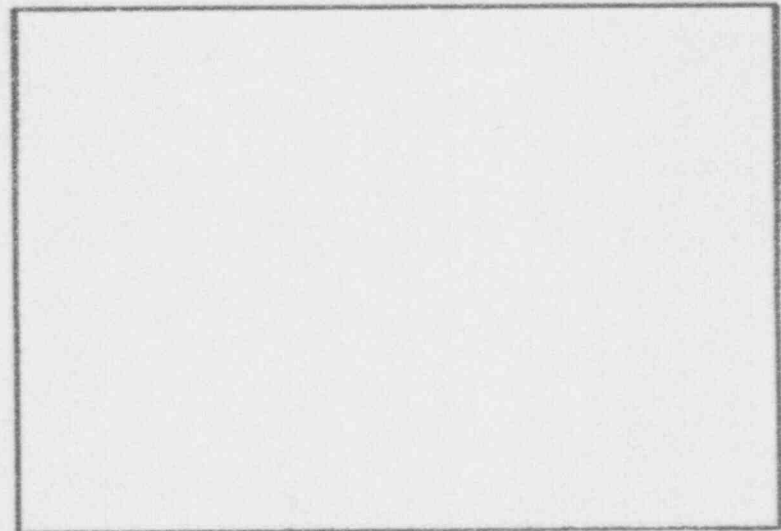
- Drawing changes
- Functional requirements document revision
- Precautions limitations and setpoints document revision
- Setpoint methodology revision
- FSAR revision

Design Considerations

- Technical specification revision
- Environmental qualification program revision
- USQ associated with new failure modes
- Site acceptance test scope
- Software verification and validation plan
- Assessment of plant grounding system

Maintenance Considerations

- Reduced time to perform surveillance tests
- Reduced time to calibrate dynamic functions
- Capability for testing in bypass
- Potential scaling problems with site-specific programs
- Tighter tech spec allowable values
- Potential impact to response time testing program
- New failure modes to diagnose and repair

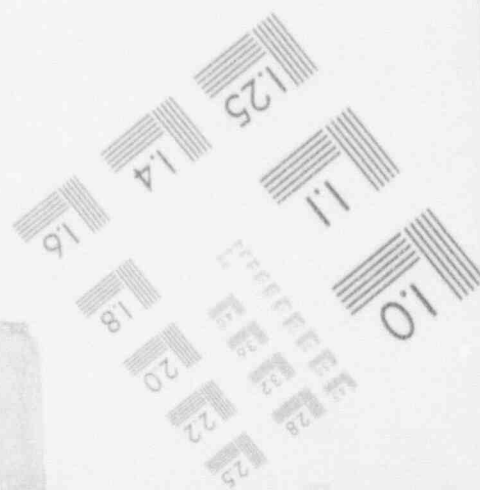
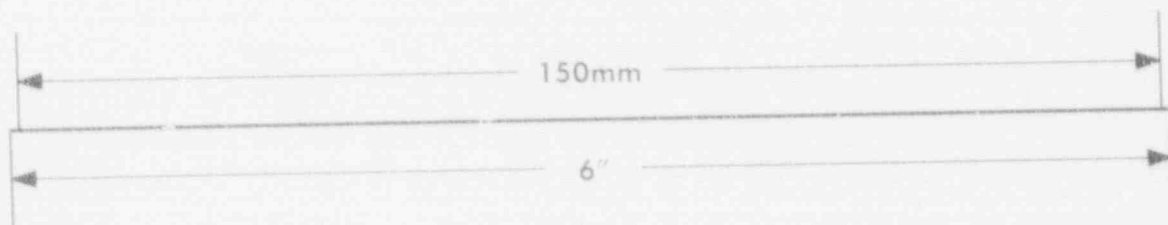
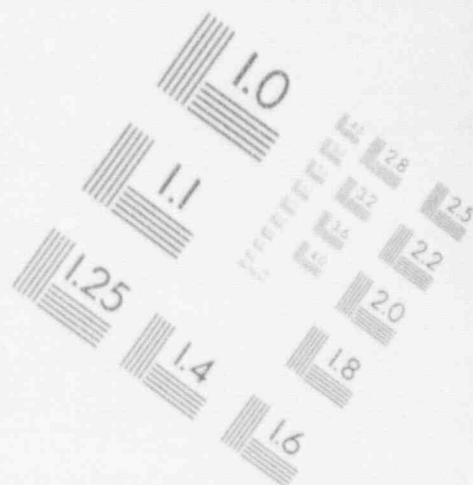
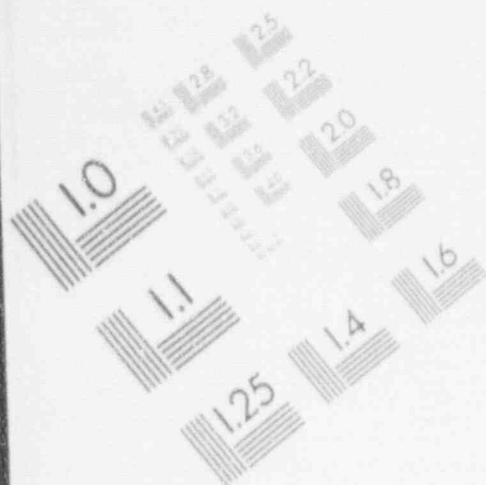


Procedures Considerations

- Revised
 - Functional tests
 - Channel calibrations
 - Response time tests
 - Channel check procedures
 - Calorimetric program

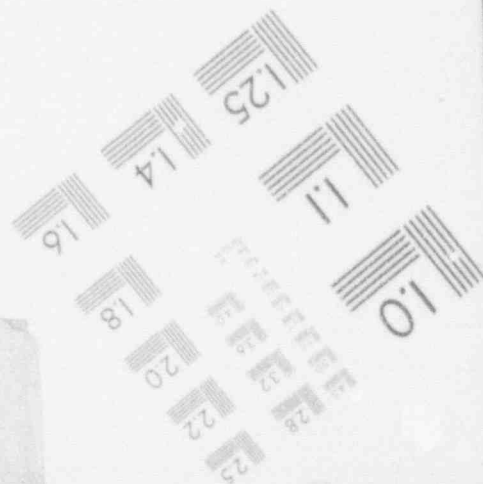
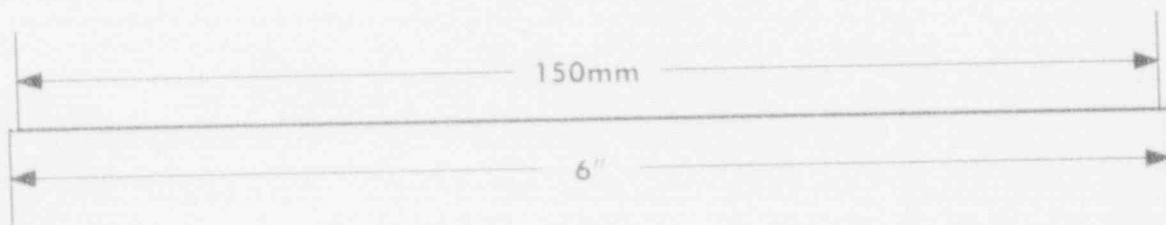
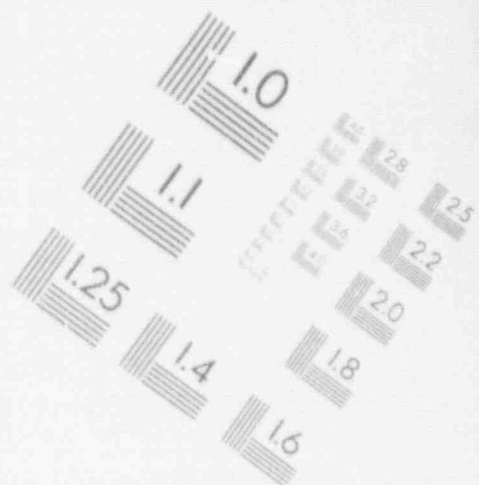
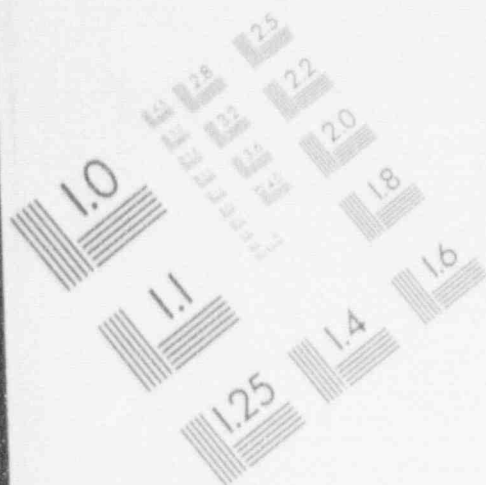
1

IMAGE EVALUATION
TEST TARGET (MT-3)



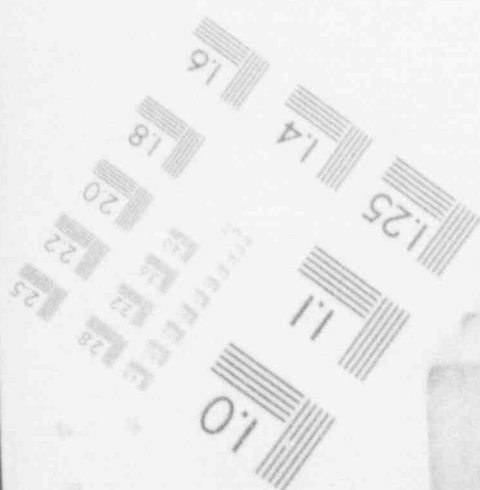
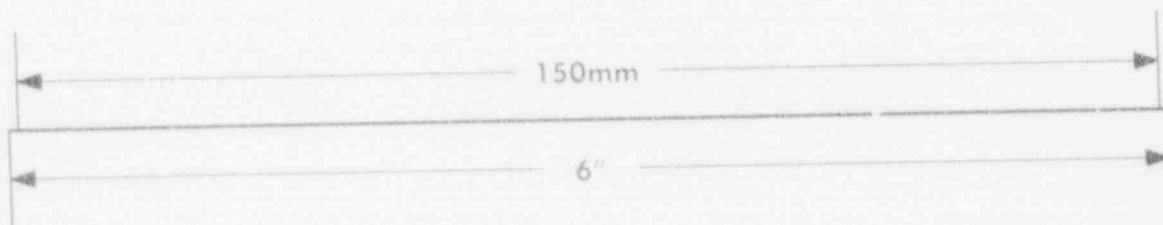
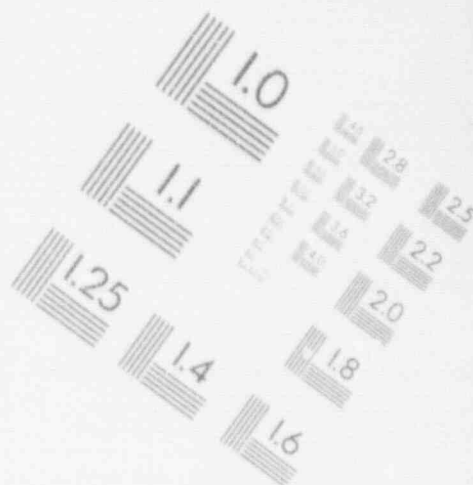
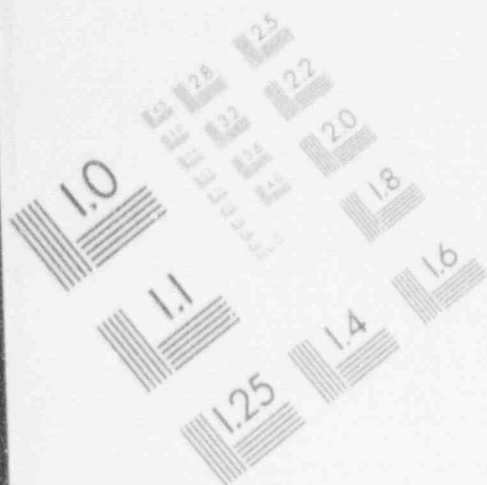
1

IMAGE EVALUATION
TEST TARGET (MT-3)



1

IMAGE EVALUATION
TEST TARGET (MT-3)



Procedures Considerations

- Revised
 - RCS flow recalibration
 - RCS RTD cross-calibration program
 - Program to remove failed channels from service
 - Annunciator response procedures
- Procedure to monitor and update RCS streaming coefficients

Operations Considerations

- New annunciators for protection system failures
- New reactor protection logic to learn
- New tech spec action statements to learn
- Significantly reduced risk of reactor trip from steam generator level during start-up and low-power operations

Training Considerations

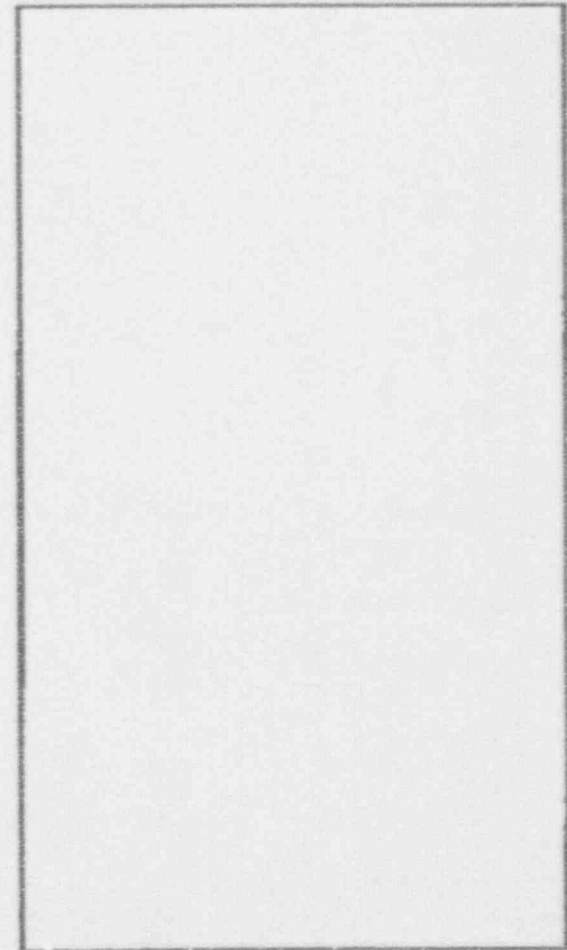
Retraining of operators and engineers

- New steam break protection
- Environmental allowance modifier and trip time delay functions
- Median signal select on steam generator-level control

Training Considerations

Training of instrument techs
on Eagle 21 hardware

- Operation of hardware
- Troubleshooting techniques
- Circuit board handling techniques
and board configuration
- Surveillance/testing procedures
- Rack parameter updating based
on site-specific data
- Use of user-configurable test
points and test carts



Lessons Learned

- Plant indicated full power Delta T changed
 - Streaming
 - Low-leakage core
- Revised method to compensate for nuclear instrumentation system gain
- Requirement to periodically measure streaming coefficients and update Eagle 21
- Requirement to periodically measure RTD current and update Eagle 21

Lessons Learned

- Environmental allowance monitor (EAM) cannot be operable during containment vent/purge on ice condenser containments
- Must go to 100% power to initially set streaming coefficients
- Must remove streaming coefficients on Delta T to perform functional tests and calibrations
- No ability to quickly download/upload tuning parameters from or into Eagle 21

Lessons Learned

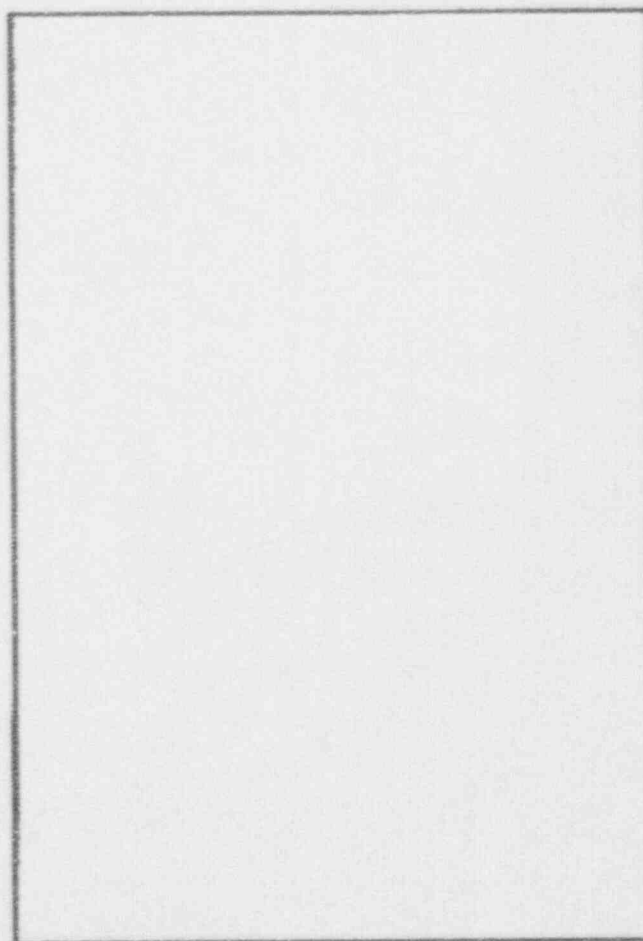
- Racks may independently monitor signals from shared sensors
 - Parameter update in two racks on streaming coefficient changes
 - RTD current changes for Delta T loops
- Cannot initiate trip from partial trip switch on energize-to-trip functions if software bypass has been established

Lessons Learned

- Cannot retrieve error messages after reset of rack
- Failure of loop processor causes digital outputs to trip but analog outputs to freeze at last value
- Racks draw more current than original analog hardware
- Connection of maintenance and test equipment to test points can cause negative spikes that can momentarily actuate bistables

Benefits of Eagle 21 at Sequoyah

- Significantly reduced calibration drift
- Reduced functional test duration
- Significantly reduced loop calibration time
- Significantly reduced time to implement rescaling of an instrument loop during testing after refueling outages
- Simplified special testing
 - Elimination of connections to each analog loop
 - Printout of data



Benefits of Eagle 21 at Sequoyah

- Eliminated need for transmitter adjustments at power-on steam flow, feedwater flow and reactor coolant flow by performing loop compensation
- Improved operator confidence in steam generator-level control operation with new setpoints and time delays provided by EAM and TTD
- Reduced vulnerability to reactor trip based on new steam break logic