A REPLY TO THE CRITICISMS OF THE KNIGHT & LEVESON EXPERIMENT

John C. Knight
University of Virginia
Charlottesville, VA 22903

Nancy G. Leveson
University of California
Irvine, CA 92717

1. Introduction

In July 1985, we presented a paper at the Fifteenth International Symposium on Fault-Tolerant Computing [KNI85] describing the results of an experiment that we performed examining an hypothesis about one aspect of *N*-version programming, i.e., the statistical independence of version failure. A longer journal paper on that research appeared in the IEEE Transactions on Software Engineering in January 1986 [KNI86].

Since our original paper appeared, some proponents of *N*-version programming have criticized us and our papers, making inaccurate statements about what we have done and what we have concluded. We have spoken and written to them privately attempting to explain their misunderstandings about our work. Unfortunately subsequent papers and public pronouncements by these individuals have contained the same misrepresentations.

We have not previously responded publicly to this criticism because we feel that our papers stand up for themselves, and we did not want to fan the flames. However, it has now been nearly 5 years since our work first appeared in print and, in our opinion, the attacks are getting more frequent, more outrageous, and farther from the truth. We have decided that it is now necessary to respond publicly to ensure that those hearing these statements do not think we are being silent because we agree with the things being said about us and our work. None of the papers criticizing our experiment have appeared in a refereed journal or been presented in a forum in which we could respond. Therefore, we are using this forum.

Nearly all of the criticism of our work has come from Professor Algirdas Avizienis of UCLA or his former students John Kelly, Michael Lyu, and Mark Joseph. We reply to their criticism by quoting some of the erroneous statements from their papers and addressing each of these statements in turn. The quotations from these papers are printed in italics in this paper.

For those who are unfamiliar with the controversy, we provide some background. An *N*-version system attempts to incorporate fault tolerance into software by executing multiple versions of a program that have been prepared independently. Their outputs are collected and examined by a decision function that chooses the output to be used by the system. For example, if the outputs are not identical but should be, the decision function might choose the majority value if there is one.

It has been suggested that the use of this technique may result in highly reliable software, even if the software versions have not been subjected to extensive testing. For example, Avizienis states:

> *By combining software versions that have not been subjected to V&V [verification and validation] testing to produce highly reliable multiversion software, we may be able to decrease cost while increasing reliability.* [AVI84]
>
> *The higher initial cost may be balanced by significant gains, such as faster release of trustworthy software, less investment and criticality in verification and validation, ...* [AVI89]

The primary argument for the attainment of ultra-high reliability using this technique is given by Avizienis:

> *It is the fundamental conjecture*† *of the NVP [N-Version Programming] approach that the independence of*

† "Conjecture" is defined by Webster's New World Dictionary to be "an inference, theory, or prediction based on guesswork".

programming efforts will assure a low probability that residual software design faults will lead to an erroneous decision by causing similar errors to occur at the same c[ross]c[heck]-points in two or more versions... The effectiveness of the NVP approach depends on the validity of this conjecture... [AVI85b]

Since the versions are written independently, it is hypothesized that they are not likely to contain the same errors, i.e., that errors in their results are uncorrelated. [AVI85a]

As Avizienis notes, this hypothesis is important because the degree to which it holds will determine the amount of reliability improvement that is realized. Eckhardt and Lee [ECK85] have shown that even small probabilities of correlated failures, i.e., deviation from statistically independent failures, cause a substantial reduction in potential reliability improvement.

The phrases ''low probability'' and ''not likely'' used by Avizienis are not quantified, and his conjecture and hypothesis are, therefore, not testable. However, statistical independence, i.e., uncorrelated failures, is well defined, implied by much of the discussion about the technique, and assumed by some practitioners [MAR83, YOU85]. In order to test for statistical independence, we designed and executed an experiment in which 27 versions of a program were prepared independently from the same requirements specification by graduate and senior undergraduate students at two universities. The students tested the programs themselves, but each program was subjected to an acceptance procedure for the experiment consisting of 200 typical inputs. Operational usage of the programs was simulated by executing them on one million inputs that were generated according to a realistic operational profile for the application. Using a statistical hypothesis test, we concluded that the assumption of independence of failures did not hold for our programs and, therefore, that reliability improvement predictions using models based on this assumption may be unrealistically optimistic. This was *all* that we concluded (see below).

The basic argument made by our critics is that if we had just used a different methodology for producing the programs, our results would have been different. For example, Avizienis states:

It is our conjecture that a rigorous application of the design paradigm, as described in this paper would have led to the elimination of most faults described in [KNI86] before acceptance of the programs. [AVI87, AVI88]

It is easy to assert that changes in experimental procedures would yield different results. Such conjectures, however, need to be supported with scientific proof before they can be accepted. Professor Avizienis and former students use two arguments to support their conjecture. Their first argument is that, in their experiments, they did not get the same results that we did:

An important conjecture of the design diversity approach is that the independence of the development process will minimize the probability that software design faults will cause similar errors to occur. Although early experiments caused concern that this assumption may not hold [KNI86], additional empirical work has shown that this assumption can hold if a 'best practice' development paradigm is used [BIS85, AVI87]. [KEL89]

These results [of the UCLA/H experiment] are different from previously published results by Knight and Leveson. [AVI87, AVI88]

Their second argument is that the claimed difference in results is accounted for by the significant differences between their experiments and ours and the inadequacies of our software development method:

[The Knight/Leveson and Scott, et al. studies] fail to recognize that NVP is a rigorous process of software development. The papers do not document the rules of isolation, and the C[ommunication]&D[ocumentation] protocol ... that are indicators of NVP quality. The V-specs of [KNI86] do not show the essential NVS [N-Version Software] attributes. It must be concluded that the authors are assessing their own ad hoc processes for writing multiple programs, rather than the NVP process as developed at UCLA, and that their numerical results uniquely take the measure of the quality of their casual programming process and their classroom programmers. The claims that the NVP process was investigated are not supported by the documentation of the software development process. [AVI89]

In summary, we have reviewed the V/UCI experiment and found reasons that may account for this outcome: the small scale and limited diversity potential of the experiment, lack of MVS [Multi-Version Software] software development disciplines, and apparently inadequate testing and processing of MVS systems. [AVI88]

Both of these arguments are unfounded. We examine each of these in turn.

2. Different Results

The first claim by our critics is that their results are different from ours. For the benefit of the reader, we repeat the conclusion we drew in our 1986 paper [KNI86]:

> ''For the particular problem that was programmed for this experiment, we conclude that the assumption of independence of errors that is fundamental to some analyses of *N*-version programming *does not hold*. Using a probabilistic model based on independence, our results indicate that the model has to be rejected at the 99% confidence level.

This was our only conclusion.

Table 1 shows data from other relevant studies that have been conducted. Chen generated 16 programs, chose 4 of these to consider further, and added 3 programs written by ''the authors'' for a total of 7 programs written in PL/I [CHE78]. Kelly used three different specifications, written in OBJ, PDL, and English, to generate 18 programs which he executed on 100 input cases [KEL83, AVI84]. Kelly was also involved in a team effort headed by NASA involving 4 universities (UCSB, Virginia, Illinois, and NCSU) in which 20 programs were generated from a single specification [ECK89]. We refer to this experiment here as the NASA experiment. Avizienis, Lyu, and Schutz generated 6 programs written in 6 languages [AVI87] from a single specification. We refer to this experiment as the UCLA/H experiment.

	Knight/Leveson	Chen	Kelly	NASA	UCLA/H
Number of versions.	27	7	18	20	6
Average no. of faults per post- development version.	1.6	not reported	not reported	not reported	1.8
No. of simulated use input cases.	1,000,000	32	100	921,000	1000
Average individual failure rate.	.0007	not reported	.27	.006	not reported
Average 3-version failure rate reported.	.00004	.10	.20	.0002	not reported
Statistically independent failure behavior.	no	not tested	not tested	no	not tested

Table 1 - Data From Relevant Experiments

Joseph makes the following claim:

> *Several experiments on NVP performed at UCLA [AVI84, AVI87] have not discovered the high rates of failure as reported in Knight/Leveson.* [JOS88]

There are two types of failure rates to which Joseph may be referring: individual version failure rates and 3-version failure rates. Failure rate is calculated by dividing the number of failures by the number of input cases. As shown in the table, both the individual and the 3-version failure rates are much lower for our experiment than for any of the UCLA experiments for which the data was collected and published.

The comparison with the Kelly experiment may be unfair because his data was not generated randomly and may have been generated to check for difficult cases. However, at best, the rates are uncomparable. They certainly are not better than ours.

Avizienis and Lyu do not report failure rates in any of the papers published by them. They report only the number of faults found. It is not possible to hypothesize failure rates during execution from the number of faults removed during testing. However, it might be noted that the average number of faults per program that were detected in simulated use of their programs was greater than ours. Without collecting data on failure rates, it is not possible to draw any conclusions about the effectiveness of *N*-version programming.

There have been many statements that the UCLA/H study got different results than we did. For example:

> *Although early experiments caused concern that this assumption may not hold [KNI86], additional empirical work has shown that this assumption can hold if a 'best practice' development paradigm is used [BIS85, AVI87].* [KEL89]
>
> *The comparison of the results in the V/UCI and UCLA/H studies thus shows major disagreements* [AVI88]

From the published papers, we can find no evidence that the independence assumption was tested in the UCLA/H study.

The following statement may explain some of the confusion about the UCLA/H study results:

> *A major observation that relates to the effectiveness of MVS is that similar and time-coincident errors (due to identical faults in two versions) were rare. Only one identical pair existed in the 82 faults removed from the six versions before acceptance [in the UCLA/H study]. During post-acceptance testing and inspection, five faults were uncovered by testing. One pair again was identical. Six more faults were discovered by code inspection, all unrelated and different ... These results are different from previously published results by Knight and Leveson.* [AVI88]

The relevant factor is not whether the faults are identical, but whether failures are coincident. When attempting to provide fault tolerance through voting, it is the intermediate results or outputs that are compared, not the faults — looking at the faults in a program does not provide information about the failure behavior of the executing program. The UCLA/H results cannot be compared with ours because the relevant analysis is not reported; independence is a statistical property and must be tested statistically.

In fact, faults do not need to be identical to produce statistically-dependent coincident failures. Many of the faults in our programs that produced such failures were seemingly unrelated on the surface, sometimes occurring in totally unrelated parts of the programs. This puzzled us until we realized that the relationship is in the functions computed by the paths taken for that input rather than in any particular ''faulty'' statements on that path. In a paper to be published in the February 1990 issue of IEEE Transactions on Software Engineering, we describe the faults in our programs and provide a model that explains why programs fail coincidentally due to faults that are not identical.

Avizienis refers to our study and another one that got similar results [SCO87] and states:

These efforts serve to illustrate the pitfalls of premature preoccupation with numerical results. [AVI89]

Without looking at numerical results, one cannot do the necessary statistical analysis to determine whether the independence hypothesis holds or determine what type of benefits can be expected from using *N*-version programming. It is surprising that Professor Avizienis believes that it is too early to look at numerical results but from his statements does not seem to believe that it is too early to use this technique in safety-critical applications such as commercial aircraft [AVI87, AVI88, AVI89, etc.].

Avizienis makes the following truly outrageous statement when describing our work:

The use of the term "experiment" is misleading, since it implies repeatability of the experimental procedure that is taken for granted in science. [AVI89]

Our experimental procedure is completely repeatable. We published precisely what we did and the requirements specification we used. Any researcher could have repeated our experiment. In fact, the result has been confirmed. Using the NASA programs, which were developed using a method closely related to that used in the UCLA/H study, a group (including Dr. Kelly) led by Dr. David Eckhardt of NASA's Langley Research Center collected the same type of data and came to the same conclusion that we did. [ECK89].

In summary, the claims that our critics did not get our results are unsupported and appear to be based more on wishful thinking than scientific analysis.

3. Methodological Comparisons

The second set of criticisms of our experiment have to do with the software development method that we used. For example, Joseph states:

It would be a mistake to accept the Knight and Leveson work at face value without considering its many weaknesses. It is proposed that the study did not use NVP due to inadequacies in proper system development methods. [JOS88]

and Avizienis states:

The claims that the NVP process was investigated are not supported by the documentation of the software development process in [Knight/Leveson]. [AVI89]

We used the methodology used by Chen [CHE78] and Kelly [KEL83, AVI84] and followed exactly what was stated by Avizienis in [AVI77, AVI85b]. According to these definitions, *N*-version programming means writing *N* versions independently. This is what we did. To claim that we did not use *N*-version programming is ridiculous.

The only significant difference between the version development methodology we used and that used in the UCLA/H experiment (which occurred 3 years after ours) is that our methodology is more likely to result in design diversity. The current ''paradigm'' promoted by Avizienis and Lyu [AVI88] involves overspecifying the design and thus limiting potential diversity.

The specific criticisms in the area of methodology that have been leveled at us have to do with quality of versions, testing, voting procedure, diversity and scale, specification, communication and isolation of programmers, and programming effort involved. We examine each in turn.

3.1. Quality of Versions

Avizienis and Joseph state:

[The Knight and Leveson] numerical results uniquely take the measure of the quality of their casual programming process. [AVI89]

NVP does not mean low quality versions. In [KNI86], no software development standards or methods were required of the programmers. This is an essential requirement for all development and raises doubts about the quality of the generated versions. [JOS88]

There is a misunderstanding here about what we said in our paper. First of all, we did not say that no software development methods were used. We said that no one particular method was *imposed* on all the programmers. All of our participants were graduate students in software engineering or related fields or were undergraduates taking a senior-level, advanced software engineering course.

We are particularly concerned about the claim that the programs are low quality despite the evidence to the contrary in our papers. Six of the twenty-seven programs did not fail on any of the million input cases. The average failure probability was 0.0007 and the worst was 0.009. This should be compared to the published UCLA experiments [CHE78], [KEL83], and [AVI84], which all had much poorer quality versions than we did in our experiment. No failure probabilities have been published for the UCLA/H experiment but the reported number of faults per version is comparable with ours.

Our reliability is of the same order as that achieved in industrial settings and is better than that of the studies by our critics. We can see no possible basis for an argument that our versions are low quality.

3.2. Testing

Avizienis, Lyu, and Joseph state:

In summary, we have reviewed the V/UCI experiment and found reasons that may account for this outcome: ... apparently inadequate testing and processing of MVS versions. [AVI88]

Acceptance tests for each version were too small (i.e., only 200 test cases). Also, operational testing used randomly generated test cases. For critical and life-critical computer systems this is completely unacceptable. [JOS88]

This has been one of the frequent criticisms about our experiment. The critics have refused to believe our statements (both public and private, oral and written) that they are misinformed. The programs were tested by their authors before they were submitted to the acceptance procedure. The acceptance procedure was an experimental artifact, not part of any software development process. The purpose was merely to ensure that the versions were all suitable for the experiment before the programmers became unavailable. We purposely did not want to put too many input cases into the acceptance procedure in order not to bias the experiment by finding and eliminating faults outside the experimental domain. Different inputs were used in the acceptance procedure for each version to avoid filtering common faults. As it turned out, very few of the programs failed the acceptance test (and if they did, it was usually only one fault involved).

As we said above, the programs were all of very high quality so they were obviously tested by the developers. What Joseph calls ''operational testing'' was not testing. We said in our papers that it was simulated use of the programs. We were trying to simulate the lifetime production use of the software. That is why we used inputs that were randomly generated. Note, however, that they were randomly generated according to what the people at Boeing felt was a realistic operational profile for this application [NAG82].

We agree, of course, that the procedures we followed were not sufficiently complete for life-critical software. We never claimed that they were. If in their statements Avizienis, Lyu, and Joseph are implying that the performance of *N*-version programming would be better using a different development methodology, then this needs to be shown using controlled experiments to demonstrate that there is a statistically significant difference in the number of correlated failures under different development procedures for the same application. This cannot be assumed without experimental data of which there is currently none.

But the proof is really ‘‘in the pudding,’’ as they say. Our higher quality programs seem to imply that our testing was as adequate as that in any of the experiments of our critics.

Kelly also claims that our testing was inadequate and that this explains our results:

> *Key results from empirical studies addressing the similar error problem include: ... (c) Versions which have not undergone systematic testing contain related faults [KNI86, SCO87].* [KEL89]

The implication in this statement is that versions that have undergone systematic testing will not contain related faults. However, we note that all experiments on *N*-version programming that we know about have found related faults, no matter what type of testing they have undergone. Shimeall and Leveson [SHI88], in a study that compared fault elimination and fault tolerance methods using eight software versions, found that extensive testing was not likely to find the faults that resulted in correlated failures. This makes some sense intuitively since there is reason to believe that if these common errors are likely to be made by the programmers, they are also likely to be made by those constructing test data. There is no scientific data to show that testing of any kind has an effect on related faults. We note that in the UCLA/H study a much higher percentage of the faults found after testing were ‘‘identical’’ than those found during testing.

3.3. Voting Procedure

In discussing our definition of failure, Joseph states:

> *The definition of complete NVP failure is incorrect. In [KNI86] an NVP system fails if a majority of versions fail at the same time, regardless of whether the errors produced were similar. An NVP system will produce the wrong result only if similar, coincident errors are generated.* [JOS88]

The author’s statement implies that a program that does not produce an output when required has not failed; a program only fails when it produces incorrect output. This differs from every definition of failure we have seen. For example, ANSI/IEEE Standard 729-1983 defines a failure as the inability of a system or system component to perform a required function within specified limits.

If we are prepared to accept that producing no result is satisfactory performance, then any program can be made arbitrarily reliable by making it abort on every execution. The purpose of fault masking in *N*-version systems is to continue to provide service in the presence of faults. This will not be possible if a majority of versions fail no matter how they fail.

In [KEL86] it was stated that we took an extreme position by using vector voting. We have revoted the programs using element-by-element voting, as was suggested, and the results are identical to the ones we published originally [MAR87].

Avizienis and Lyu state (the emphasis is theirs):

> *In the testing and processing of the MVS systems, the failure detection and granularity for the V/UCI experiment was coarse, since it used one Boolean variable to represent 241 Boolean conditions for a “missile launching decision.” The UCLA/H experiment employed real number comparisons for inexact matching with specified tolerances.* [AVI88]

We used all 241 boolean conditions in the voting as is clearly stated in our paper [KNI86, page 99].

3.4. Diversity and Scale

Avizienis states:

> *The scale of the problem (and the potential for diversity) [in Knight and Leveson's experiment] is smaller [than the UCLA/H study].* [AVI88]

and:

> *We can see that the V/UCI experiment was of rather small scale, since the specification was 6 pages long and could be programmed in 327 lines. The scale of the UCLA/H experiment with 64 pages of specification and at least 1250 lines of code was significantly larger.* [AVI88]

Our Pascal programs ranged in size from 310 to 781 lines of code with an average of 554 lines (they did not include any I/O statements). The UCLA/H Pascal program was 1288 lines of code with 491 executable statements (including I/O statements). In software engineering, a medium size program is 50,000-100,000 lines of code and a large program is 500,000 to 1,000,000 lines of code. By these realistic standards, there was no difference in the scale of the two applications — both were small-scale.

The argument about the size of the specification is more interesting, however, because it addresses the more important claim that our application had limited potential for diversity because our specification was short. Note that Avizienis acknowledges in the following the potential problem of overspecification when attempting to get diverse programs from a single specification:

> *The specification of the member versions, to be called V-spec, represents the starting point of the NVP process. As such, the V-spec needs to state the functional requirements completely and unambiguously, while leaving the widest possible choice of implementations to the N programming efforts.... Such specific suggestions of "how" reduce the chances for diversity among the versions and should be systematically eliminated from the V-spec.* [AVI89]

The introduction of a new term, V-spec, clouds the issues. Usual software engineering terminology includes requirements specifications, high-level design specifications, detailed design specifications, etc; each contain different amounts of information about "how." The specification we gave our programmers was short because we purposely eliminated all information about algorithms and implementation in order to provide the maximum opportunity for diversity, i.e., it was a pure requirements specification. Since there is no precise definition of diversity, it is impossible to determine whether diversity is present or not in any set of program versions. We note, however, that informally and in our opinion, our versions were very different: they differed in program structure, algorithms, variables and data structures, length, and layout.

The reason that the specifications for the UCLA experiments have been so long is that they contained design information. For example, one of the specifications for the Kelly programs, which averaged 300 lines of PL/I code, was a 73-page specification written in PDL (Program Design Language). In the UCLA/H study, the specification was 64 pages long and included detailed information about the algorithms, variables, and data structures to be used. Here are some quotes from their description of the results of the UCLA/H study (emphasis is ours):

> *Primitive operations are integrators, linear filters, magnitude limiters, and rate limiters. The algorithms for these operations were <u>exactly specified,</u> however, different choices of which primitive operations to implement as subprograms have been made, mainly whether the integrators include limits on the magnitude of the output value (as is required in most cases), or not.* [AVI87]

> *Two factors that limit actual diversity have been observed in the course of this assessment... <u>algorithms specified by figures</u> were generally implemented by following the corresponding figure from top to bottom... In retrospect, a second reason for this lack of diversity is that we have concluded that the <u>logic part</u> of the Logic Mode was <u>overspecified.</u>* [AVI87]

The H[oneywell]/S[perry] concept of "test points" is the second factor that tends to limit diversity. Their purpose is to output and compare not only the final result of the major subfunctions, but also some intermediate results. However, that restricted the programmers on their choices of which primitive operations to combine (efficiently!) into one programming language statement. In effect, the intermediate values to be computed were chosen for them. [AVI87]

In their post-assessment of the diversity of the versions [AVI87] obtained from this detailed design specification, the major differences seemed to be syntactic rather than semantic, e.g., the use of parameter passing vs. global variables, differences in the calling structure of the procedures, the use of subprograms vs. functions. None of these seem to us to be very significant in terms of providing fault tolerance of design errors.

Another source of their lack of diversity stems from the use of cross-checks points to vote on intermediate results and what they call Community Error Recovery [TSO87]:

[The cross-check points] have to be executed in a certain predetermined order, but again great care was taken not to overly restrict the possible choices of computation sequence. [AVI87]

One recovery point is used to recover a failed version by supplying it with a set of new internal state variables that are obtained from the other versions by the Community Error Recovery Technique. [AVI87]

In order to effect this type of recovery, the internal states of the versions must be identical. In fact, the versions had a disagreement caused by:

the introduction of new, unspecified state variables which we call "underground variables", since they are neither checked nor corrected in any cross-check or recovery point... A new design rule for multi-version software must be stated as "Do not introduce any 'underground' variables." [AVI87]

It is not surprising that few identical faults were found by a test procedure that involved checking the results of these programs against each other since the designs were, for all practical purposes, identical.

On a related issue, Avizienis states in a recent paper:

The specification for simplex software tend to contain guidance not only "what" needs to be done, but also "how" the solution ought to be approached. [AVI89]

Specifications for simplex (single-version) software are no different in this respect than for *N*-version software. In fact, the same specifications can be used for both types of software development, and requirements specifications for simplex software can just as easily contain only "what" as requirements specifications for multi-version software. To the contrary, we have found that in practice (and in the UCLA experiments) specifications for *N*-version systems almost always require specification of the "how" (i.e., detailed design) in order to provide the kinds of voting and comparison required for *N*-version programming. Avizienis, describes these difficulties with respect to the versions in his UCLA/H experiment:

It was decided that different algorithms were not suitable for the scope of FCCs [flight control computers] due to potential timing problems and difficulties in proving their correctness (guaranteed matching among them). [AVI87]

In real-life systems that use this technique, differences between the supposedly "diverse" modules are often minor. In fact, our experiment was unique and unrealistic in that we allowed more diversity than is usual for these systems in practice (or in the other studies that have been done), and thus our programs are more likely to provide design fault tolerance. This is rather frightening since this real-life software often is depended on for safety-critical activities.

3.5. Specification

Avizienis states:

> *The V-specs [of Knight and Leveson] do not show the essential NVS attributes.* [AVI89]

No explanation of what attributes are missing is given, and we are at a loss to figure out what they could be. Avizienis' only specific criticism of our specification seems to be one of length. The primary difference between our specification and those of his latest UCLA/H study is that our specifications are closer to what he describes as necessary for *N*-version programming, i.e., they specify only "what" without "how."

3.6. Communication and Isolation

Avizienis states:

> *There was no required communication protocol for the programmers in the V/UCI experiment, while the communication protocol for the UCLA/H experiment was well-defined, and rigorously enforced.* [AVI88]
>
> *The papers [describing Knight and Leveson's experiment] do not document the rules of isolation, and the C&D protocol that are indicators of NVP quality.* [AVI89]

The communication protocols for the two experiments are identical. Our protocol is documented in our paper [KNI86, page 98]. Isolation, which is also documented in the same paper [KNI86], was enforced in the same way as in the UCLA experiment with the added factor that some of the programmers were separated by 3000 miles and were unknown to each other. All correlated failures involved versions from both schools.

3.7. Programming Time

Avizienis states:

> *The V/UCI experiment was a class project during one Quarter term, and each student produced the program alone. On the contrary, programmers in the UCLA/H experiment worked in two-member teams and were paid a full-time research assistant (RA) salary during a class-free 12 week period during the summer.* [AVI88]

The real problem is that students were used as opposed to professional programmers in all of these experiments, including ours. Avizienis argues that his students are somehow less likely to make similar errors because they were paid, because they worked during the summer and not during the academic year, and because they worked two weeks longer. We cannot see how any of these things could make a difference.

There is likely to be some difference in results if professional programmers are involved. But it is not intuitively obvious that fewer coincident failures would be the result. It seems reasonable to hypothesize that student programs would be prone to more randomness in errors and designs than those written by professionals. It is interesting to note, however, that one UCI participant in our study, who had more than ten years of professional scientific programming experience, had faults that correlated in failure characteristics with a UVA student who had no professional programming experience.

4. Conclusions

Joseph states:

> *Thus, [the Knight and Leveson] results are misleading and should not be used by themselves as a basis for a decision about the effectiveness of NVP. Real world experience, not a class room assignment as in [KNI86], is needed. Currently, several systems in Europe are using NVP (e.g., the European designed Airbus 320*

[ROU86] [AVW87]). [JOS88]

Our results are not misleading. Our conclusion is simple and clearly stated. We repeat again part of the conclusion section from our paper [KNI86] (italics are from the original paper):

> "For the particular problem that was programmed for this experiment, we conclude that the assumption of independence of errors that is fundamental to some analyses of *N*-version programming *does not hold*. Using a probabilistic model based on independence, our results indicate that the model has to be rejected at the 99% confidence level.
>
> "It is important to understand the meaning of this statement. First, it is conditional *on the application that we used*. The result may or may not extend to other programs, we do not know. Other experiments must be carried out to gather data similar to ours in order to be able to draw *general* conclusions. However, the result does suggest that the use of *N*-version programming in crucial systems should be deferred until further evidence is available if the reliability analysis of the system has been performed assuming independence of failures."

We never suggested that our result should be used by itself as a basis for a decision about the effectiveness of *N*-version programming. We merely suggested that caution would be appropriate. We feel strongly that careful, controlled experimentation in a realistic environment is required. However, advocating careful laboratory experimentation is different from advocating the accumulation of experience by using the technique in real, safety-critical applications where loss of life is possible, such as commercial aircraft [JOS88].

Attacking one experiment by John Knight and Nancy Leveson does not make *N*-version programming a reasonable way to ensure the safety of software. All of the university experiments, including ours, have been limited in one way or another. None (except Shimeall and Leveson [SHI88]) has attempted to compare *N*-version programming with the alternatives to find out whether the money and resources could have been more effectively spent on other techniques such as sophisticated testing or formal verification.

We did not say that *N*-version programming should not be used although it would not be our first choice of techniques for increasing reliability and safety. We just do not believe that it should be relied upon to provide ultra-high reliability. The FAA requires that failures of critical systems in commercial air transports be "extremely improbable". The phrase "extremely improbable" is defined by the FAA as "not expected to occur within the total life span of the whole fleet of the model [WAT78]." In practice, where such reliability can be analyzed, the phrase is taken to mean no more than 10^{-9} failures per hour of operation or per event for activities such as landing. No one has demonstrated that *N*-version programming can guarantee achievement of this level of reliability. In fact, we know only of counterexamples.

Our conclusion is modest and follows from the experimental data. Our research is not "flawed" as it has been described in public by our critics at professional meetings. Our critics should refrain from attributing conclusions to us that we have not drawn, from making statements about the quality or diversity of our programs without the data to substantiate these judgements, and from making unsupported comparisons of our results with theirs.

Until *N*-version programming has been shown to achieve ultra-high reliability and/or has been shown to achieve higher reliability than alternative ways of building software, the claims that it does so should be considered unproven hypotheses. Until these hypotheses are shown to hold for controlled experiments, depending on *N*-version programming in real systems to achieve ultra-high reliability where people's lives are at risk seems to us to raise important ethical and moral questions. Attacking us or our papers will not change this.

REFERENCES

[AVI77] A. Avizienis and L. Chen, “On the Implementation of *N*-Version Programming for Software Fault-Tolerance During Program Execution”, *Proc. of Compsac ’77*, November 1977, pp. 149-155.

[AVI84] A. Avizienis and J.P.J. Kelly, “Fault Tolerance by Design Diversity: Concepts and Experiments”, *IEEE Computer*, Vol. 17, No. 8, August 1984, pp. 67-80.

[AVI85a] A. Avizienis, *et al.*, “The UCLA Dedix System: A Distributed Testbed for Multiple-Version Software”, *15th Int. Symposium on Fault-Tolerant Computing*, Michigan, June 1985, pp. 126-134.

[AVI85b] A. Avizienis, “The *N*-Version Approach to Fault-Tolerant Software”, *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 12, December 1985, pp. 1491-1501

[AVI87] A. Avizienis, M.R. Lyu, and W. Schutz, “In Search of Effective Diversity: A Six-Language Study of Fault-Tolerant Control Software”, Tech. Report CSD-870060, UCLA, November 1987.

[AVI88] A. Avizienis and M.R. Lyu, “On the Effectiveness of Multiversion Software in Digital Avionics”, *AIAA/IEEE 8th Digital Avionics Systems Conference*, San Jose, October 1988, pp. 422-427.

[AVI89] A. Avizienis, “Software Fault Tolerance”, *IFIP XI World Computer Congress ’89*, San Francisco, August 1989.

[AVW87] “Airbus 320, the New Generation Aircraft”, *Aviation Week & Space Technology*, February 2, 1987, pp. 45-66.

[BIS85] P.G. Bishop, *et al.*, “Project on Diverse Software — An Experiment in Software Reliability”, *Proceedings IFAC Workshop Safecomp ’85*, Como, Italy 1985.

[CHE78] L. Chen and A A. Avizienis, “*N*-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation”, *Digest FTCS-8: Eighth International Symposium on Fault-Tolerant Computing*, Tolouse, France, June 1978, pp 3-9.

[ECK85] D.E. Eckhardt and L.D. Lee, “A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors”, *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 12, December 1985, pp. 1511-1516.

[ECK89] D.E. Eckhardt, *et al.*, “An Experimental Evaluation of Software Redundancy as a Strategy for Improving Reliability”, Technical Report, NASA/Langley Research Center, submitted for publication.

[JOS88] M.K. Joseph, “Architectural Issues in Fault-Tolerant, Secure Computing Systems”, Ph.D. Dissertation, Dept. of Computer Science, UCLA, 1988.

[KEL83] J.P.J. Kelly, and A. Avizienis, “A Specification-Oriented Multi-Version Software Experiment”, *Proc. 13th International Symposium on Fault-Tolerant Computing*, Milan, Italy, June 1983, pp. 120-126.

[KEL86] J.P.J. Kelly, *et al.*, “Multi-Version Software Development”, *Proc. Safecomp ’86*, Sarlat, France, October 1986, pp. 43-49

[KEL89] J.P.J. Kelly, “Current Experiences with Fault Tolerant Software Design: Dependability Through Diverse Formal Specifications”, *Conference on Fault-Tolerant Computing Systems*, Germany, September 1989, pp. 134-149.

[KNI85] J.C. Knight and N.G. Leveson, “A Large Scale Experiment In *N*-Version Programming”, Digest of Papers FTCS-15: *Fifteenth International Symposium on Fault-Tolerant Computing*, June 1985, Ann Arbor, MI. pp. 135-139.

[KNI86] J.C. Knight and N.G. Leveson, “An Experimental Evaluation of the Assumption of Independence in Multi-version Programming”, *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1 (January 1986), pp. 96-109.

[MAR87] A.J. Margosis, “Empirical Studies of Multi-Version System Performance”, Master’s Thesis, University of Virginia, January 1988.

[MAR83] D.J. Martin, “Dissimilar software in high integrity applications in flight controls”, *Software for Avionics*, AGARD Conference Proceedings, No. 330, pp. 36-1 to 36-9, January 1983.

[NAG82] P.M. Nagel and J.A. Skrivan, “Software Reliability: Repetitive Run Experimentation and Modelling”, Technical Report NASA CR-165836, NASA/Langley Research Center, February 1982.

[ROU86] J.C. Rouquet and P.J. Traverse, "Safe and Reliable Computing on Board the Airbus and ATR Aircraft", *Proc. SAFECOMP '86*, Sarlat, France, October 1986, pp. 93-97.

[SCO87] R.K. Scott, J.W. Gault, and D.F. McAllister, "Fault-Tolerant Software Reliability Modeling", *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 5, May 1987, pp. 582-592.

[SHI88] T.J. Shimeall and N.G. Leveson, "An Empirical Comparison of Software Fault Tolerance and Fault Elimination", *Proc. 2nd Workshop on Software Testing, Verification, and Analysis*, Banff, July 1988. (A more complete description is available as Tech. Report NPS52-89-047, Naval Postgraduate School, July 1989.

[TSO87] K.S. Tso and A. Avizienis, "Community Error Recovery in N-Version Software: A Design Study with Experimentation", *Digest 17th Int. Symposium on Fault-Tolerant Computing*, Pittsburgh, July 1987, pp. 127-133.

[WAT78] H.E. Waterman, "FAA's Certification Position on Advanced Avionics", *AIAA Astronautics and Aeronautics*, May 1978, pp. 49-51.

[YOU85] L.J. Yount, *et al.*, "Fault Effect Protection and Partitioning for Fly-by-Wire/Fly-by-Light Avionics Systems", *AIAA Computer in Aerospace V Conference*, Long Beach, August 1985.