Knight and N. G. Leveson, John C. (1986). “An Experimental Evaluation of the Assumption of Independence in Multiversion Programming.” IEEE Transactions on Software Engineering **12**(1): 96-109.

N-version programming has been proposed as a method of incorporating fault tolerance into software. Multiple versions of a program (i.e. 'N') are prepared and executed in parallel. Their outputs are collected and examined by a voter, and, if they are not identical, it is assumed that the majority is correct. This method depends for its reliability improvement on the assumption that programs that have been developed independently will fail independently. An experiment is described in which the fundamental axiom is tested. In all, 27 versions of a program were prepared independently from the same specification at two universities and then subjected to one million tests. The results of the tests revealed that the programs were individually extremely reliable but that the number of tests in which more than one program failed was substantially more than expected. The results of these tests are presented along with an analysis of some of the faults that were found in the programs. Background information on the programmers used is also summarized.

The Problem

Critical software applications, such as mission-critical and, in particular, safety-critical applications, have the constraint that they cannot fail. The nature of software makes this a very difficult objective to achieve in most cases. In hardware design, one approach is to introduce redundant hardware components into highly reliable hardware based systems. These redundancies of the components are taken advantage of by asking each component to do a computation. The individual results are then compared and the ultimate outcome is determined by taking a vote based on the results, with the majority winning. The premise is the failure of individual hardware components is independent of the other components in the system. That is, the failure of one component in no way influences the failure of another. These ideas have been introduced into software design in the form of *n*-version programs. The idea is that for a particular software component, *n* versions of the component will be built independently and integrated into the system along with a voting mechanism. The voting mechanism takes the output of each redundant component, compares them, and determines what the result should be based on some algorithmic voting process. Differences in the results reported by the redundant components result from some failure in one or more components. The problem with an *n*-version approach to developing software components is that their failures may not be independent. If this is true, then the results from an *n*-version software computation would give a false sense of security.

Hypothesis

n-versions of a program will not fail independently. That is, the faults made by programmers implementing each version are related and non-random.

Experimental Design

Subjects:

- 27 Graduate and Senior-level students from two different universities (9 from UVA, 18 from UCI); professional experience ranged from 0 to 10+ years; varying experience in development language (Pascal)
- 27 programs (developed by students) each implementing the same specification
- Oracle program (so-called *gold* program) - used as an oracle to test the results of each program version. Oracle program was obtained by converting an existing FORTRAN program (having demonstrated high reliability) to Pascal.
- System Specification - an anti-missile system from an aerospace company; selected because it was felt that this type of application would be the type of application requiring fault-tolerance; had been used in other software engineering experiments; required sanitizing to remove defects identified in other experiments.
- A specific Pascal compiler, running under a specific hardware/operating system was used.

Variables:

- *n*-version programs - must correctly implement specification
- Programmers - must control for skill level, knowledge, and experience.
- Oracle program - must know that it is correct.
- System Specification - must be free from defects
- Test Driver - must be implemented correctly
- Test Cases - must be randomly generated

Critique:

- There were controls over the development process itself. Students were free to use whatever technique/approach they desired.
- The proficiency of the students varied too widely. Could the common errors be the result of mistakes often made by amateurs? Ah, but this itself would prove that program faults are not independent.
- A special function that performs comparison of real-number values was provided to each student with instructions to use the function. This was necessary to avoid programs with numerical rounding.
- There was no description of what measures were taken to insure that the Pascal version of the oracle program was behaviorally identical to the FORTRAN version. Presumably, the same tests used to conclude that the FORTRAN program was highly reliable were run against the Pascal version, but this was not mentioned.

- The approach, as described, appears reasonable and sound. A question, though, is how was the development of the test driver controlled? Who did it? How were the generated values determined to be random?

Conduct of the Experiment

- Students were handed the specification and directed to develop complying software. Fifteen input data sets with expected outputs were provided to assist in unit-testing and debugging.
- Each version of the program was subjected to an acceptance test of 200 randomly generated test cases. Each acceptance test set was unique for each program.
- Programs passing acceptance the test were integrated into the collection of *n*-version programs.
- A test driver was built that generated random radar reflections and random values for all of the parameters of the problem. All 27 programs were executed against the test cases. Then, the oracle program was also executed against the test cases. The results of the individual *n*-version programs were compared against the results of the oracle program, and the result of the comparison was recorded.
- Data collected for each program execution consisted of 241 different elements. These include the input values, expected output, and state information.

Experimental Results

- Failure for a given *n*-version program was defined as a discrepancy between the 241 data values collected for that program and those values collected from the oracle program.
- Of the 27 programs, 6 reported no failure whatsoever; 21 were successful for more than 99% of the test cases; 23 out of the 27 programs were successful for more than 99.9% of the test cases.
- Several programs failed on the same test case; most common failures were where two programs failed on the same test case (551); the most extreme common failure occurred when eight programs failed on a common test case twice.
- The results of the statistical analysis were significance at $\alpha = 0.01$ for a 99% confidence interval (*z*-score was 100.55). Thus, the null hypothesis was rejected in favor of the alternate hypothesis.

Presentation

The presentation of the experiment was done very well. However, a description of the controls placed on the oracle program and test driver should have been described. Overall, a nice job.