146
35-61 P-19
219 [illegible]

Design For Validation
Based on Formal Methods

Ricky W. Butler

NASA Langley Research Center
Hampton, VA 23665

VALIDATION OF ULTRA-RELIABLE SYSTEMS

DECOMPOSES INTO TWO SUBPROBLEMS

1. Quantification of probability of system failure due to physical failure
2. Establishing that **DESIGN ERRORS** are not present [1].

[1](note. Quantification of 2 is infeasible)

Achieving Ultra-Reliable Software (Approaches)

- Testing (Lots of it)
- Design Diversity (e.g. N-version programming)
- Fault Avoidance:
  - Formal Specification/Verification
  - Automatic Program Synthesis
  - Reusable Modules

Life-Testing

Basic Observation:

10^{-9} Probability of failure estimate for a 1 hour mission

REQUIRES

$> 10^9$ hours of testing

(10^9 hours = 114,000 years)

Design Diversity

1. Separate Design/Implementation Teams
2. Same Specification
3. Multiple Versions
4. Non-exact Threshold Voters
5. Hope design flaws manifest errors independently or nearly so.

The Big Problem For Design Diversity Advocates

- experiments in the low-nominal reliability region have shown that the number of near-coincident failures far exceeds the number predicted by an independence model.

- Certainly independence cannot be assumed axiomatically for ultrareliable regime

- If cannot assume independence must measure correlations.

Quantification of N-version programs not feasible in the ultrareliable regime

- Since one cannot assume independence, it must be treated as black box
- Back to life-testing problem again
- Any alternative model would have to be validated. **But How?**

How do we get ultra-reliable numbers for hardware (physical failure)?

THE ONLY THING THAT ENABLES QUANTIFICATION OF ULTRA-RELIABILITY FOR H/W IS THE

INDEPENDENCE ASSUMPTION !!

- *THE INDEPENDENCE ASSUMPTION* ***CANNOT BE DEMONSTRAT*** *FOR MULTI-VERSION S/W IN THE ULTRA-RELIABLE REGION*

The Danger of Design Diversity (N-version Programming, Recovery Blocks, etc.)

- creates an ”illusion” of ultra-reliability. By assuming independence, the advocators of S/W fault-tolerance generate ultra-high estimates of reliability.

- As long as industry/certification agencies believe that S/W fault-tolerance will solve the problem, formal methods will not be pursued.

Design For Validation

1. Designing a system in a manner that a complete and accurate reliability model can be constructed. All parameters of the model which cannot be deduced from the logical design must be measured. All such parameters must be measurable within a feasible amount of time.

2. The design process makes tradeoffs in favor of designs which minimize the number of measurable parameters in order to reduce the validation cost. A design which has exceptional performance properties yet requires the measurement of hundreds of parameters (e.g say by time-consuming fault-injection experiments) would be rejected over a less capable system that requires minimal expermimentation.

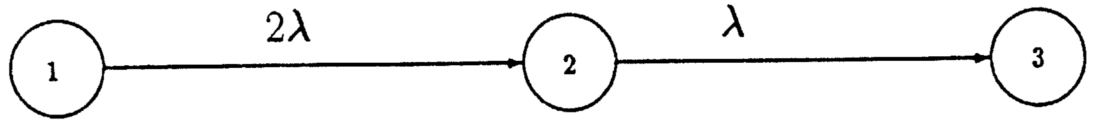
3. The system is designed in a manner that enables a proof of correctness of its logical structure. Thus, the reliability model does not include transitions representing design errors.

4. The reliability model is shown to be accurate with respect to the system implementation. This is accomplished analytically.

Illustration 1

Suppose we must design a simple fault-tolerant system with a probability of failure no greater than 2×10^{-6} whose maximum mission time is 10 hours.

- We quickly eliminate the use of a simplex processor since there is no technology which can produce a processor with this low of a failure rate.
- Thus, we begin to explore the notion of fault-tolerance. We next consider the use of redundancy—how about a dual? When the first processor fails, we will automatically switch to the other processor.



$$\begin{tikzcd} \boxed{1} \arrow[r, "2\lambda"] & \boxed{2} \arrow[r, "\lambda"] & \boxed{3} \end{tikzcd}$$

Unfortunately, our design suffers from one major problem. It is **impossible** to *prove* that any implementation behaves in accordance with this model.

The problem is that one cannot design a dual system which can detect the failure of the first processor and switch to the second 100% of the time. Thus, we must accept the fact that there is a single-point failure in our system an include it in our reliability model

SURE Run

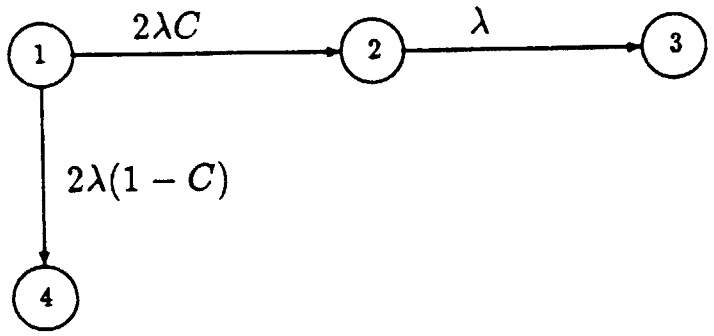
Now we have a parameter in our model which must be measured—C. It represents the fraction of single faults from which the system successfully recovers. Can this parameter can be measured in a feasible amount of time (i.e. say less than year) with statistical significance?

```
$ sure

 SURE V7.5    NASA Langley Research Center

 1? read dualspf
 2:
 3:   LAMBDA = 1E-4;
 4:   C = .9 TO 1 BY 0.01;
 5:   1,2 = 2*LAMBDA*C;
 6:   2,3 = LAMBDA;
 7:   1,4 = 2*(1-C)*LAMBDA;
 8? run

    C            LOWERBOUND    UPPERBOUND
 -----------    -----------   -----------
 9.00000e-01    2.00699e-04   2.00900e-04
 9.10000e-01    1.80729e-04   1.80910e-04
 9.20000e-01    1.60759e-04   1.60920e-04
 9.30000e-01    1.40789e-04   1.40930e-04
 9.40000e-01    1.20819e-04   1.20940e-04
 9.50000e-01    1.00849e-04   1.00950e-04
 9.60000e-01    8.08790e-05   8.09600e-05
 9.70000e-01    6.09090e-05   6.09700e-05
 9.80000e-01    4.09390e-05   4.09800e-05
 9.90000e-01    2.09690e-05   2.09900e-05
 1.00000e+00    9.99000e-07   1.00000e-06
```


$$\begin{tikzcd} 1 \arrow[r, "2\lambda C"] \arrow[d, "2\lambda(1-C)"] & 2 \arrow[r, "\lambda"] & 3 \\ 4 & & \end{tikzcd}$$

From this run we know that C must be between 9.9 and 1.0 in order to meet our reliability goal. We rerun the model to get a closer value:

2nd Run

```
$ sure

  9? read dualspf

 10:
 11:  LAMBDA = 1E-4;
 12:  C = .999 TO 1 BY 0.0001;
 13:  1,2 = 2*LAMBDA*C;
 14:  2,3 = LAMBDA;
 15:  1,4 = 2*(1-C)*LAMBDA;

      0.02 SECS. TO READ MODEL FILE
 16? run

   C            LOWERBOUND   UPPERBOUND    COMMENTS                 RUN #2
 -----------  -----------  -----------   ---------------------------------
 9.99000e-01  2.99600e-06  2.99900e-06
 9.99100e-01  2.79630e-06  2.79910e-06
 9.99200e-01  2.59660e-06  2.59920e-06
 9.99300e-01  2.39690e-06  2.39930e-06
 9.99400e-01  2.19720e-06  2.19940e-06
 9.99500e-01  1.99750e-06  1.99950e-06
 9.99600e-01  1.79780e-06  1.79960e-06
 9.99700e-01  1.59810e-06  1.59970e-06
 9.99800e-01  1.39840e-06  1.39980e-06
 9.99900e-01  1.19870e-06  1.19990e-06
 1.00000e+00  9.99000e-07  1.00000e-06
```

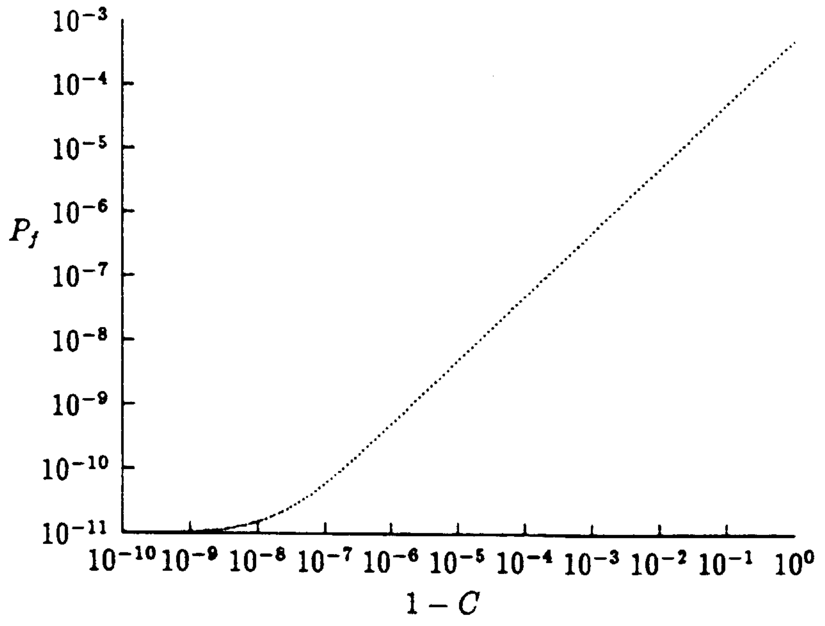
- Now, we can see that we must demonstrate that C is greater than 0.9995.
- it can shown that 20000 observations are necessary to estimate this parameter to a reasonable level of statistical significance.
- If we optimistically assume that each fault injection requires 1 minute, then this validation exercise would require 330 hours (i.e. 14 days).

In this case, we decide we can live with this amount of testing and proceed to develop our system.

Designing System with Much Higher Reliability

Now suppose we want to meet the reliability goal of $1 - 10^{-9}$.

We decide to use a nonreconfigurable 5-plex (5MR) and build a processor with a failure rate of $10^{-5}/hour$. The probability of system failure is plotted as a function of 1-C:



- The value of C must now be greater than 0.9999982.
- It can be shown that **over a million** fault injections would be required to measure this parameter even if we are very optimistic about the testing process
- If each injection required 1 minute, this would require almost 1.9 years of non-stop fault injections.

A better Way—via Design For Validation

It would be nice if we could design our system so that such an experiment is unnecessary.

- The system is designed so that a single point failure **cannot** cause system failure (i.e. C = 1).
- This is demonstrated to be true by formal proof.
- Thus, one uses the power of analysis to eliminate fault-injection style testing.

WHY FORMAL METHODS?

The successful engineering of complex computing systems will require the application of *mathematically based analysis* analogous to the structural analysis performed before a bridge or airplane wing is built.

Draft Interim Defence Standard 00-55

Quote from the foreward to the Draft Standard:

The Steering Group "has determined that the current approach which is based on system testing and oversight of the design process will, in the long-term, become cumbersome and inefficient for the assurance of the safety of increasingly sophisticated software".

"The Steering Group therefore proposes the adoption of *Formal Design Methods*, based on rigorous mathematical principles, for the implementation of safety-critical computer software".

Levels of Formal Methods

Level 0: Static Code Analysis (i.e. no semantic analysis)

Level 1: Specification using mathematical logic or language with a formal semantics (i.e. meaning expressible in logic)

Level 2: Formal Specification + Hand Proofs

Level 3: Formal Specification + Mechanical Proofs