An Evaluation of Software Fault Tolerance in a Practical System

T. Anderson *
D. N. Halliwell *
P. A. Barrett **
M. R. Moulding **

Centre for Software Reliability
University of Newcastle upon Tyne

Abstract

An experimental project to assess the effectiveness of software fault tolerance techniques is described. Techniques were developed for, and applied to, a realistic implementation of a practical real-time system, namely a naval command and control system. Reliability data was collected by running this system with a simulated tactical environment for a variety of action scenarios. Analysis of the data confirms that software fault tolerance techniques can significantly enhance system reliability.

Introduction

Over the last ten years there has been considerable research activity in the field of software fault tolerance. (See, for example, references 1-5.) An outcome of this research has been the identification of a number of techniques and tools for providing software fault tolerance, including recovery blocks [6] and N-version programming [7]. However, the use of such techniques in practical systems has not yet become widespread, although dual-software systems have been constructed for critical systems [8,9]. One reason may be the absence of evaluation studies of the techniques in practical systems. To date, evaluation of software fault tolerance has either been conducted by statistical modelling techniques [10] or by empirical studies of multiple versions of software modules [11]. Both of these approaches have their limitations. The modelling approach is often bedevilled by unjustified assumptions and/or unquantifiable parameters, whereas the empirical approach has usually had to be applied to relatively small modules (because of cost considerations). Nevertheless, both approaches have indicated the potential for significant gains in software reliability from the use of fault tolerance techniques.

This paper reports on a two and a half year project (Aug. 1981 - Feb. 1984) conducted at the University of Newcastle upon Tyne in conjunction with MARI, the Microelectronics Applications Research Institute. The aims of this project were: to refine and develop software fault tolerance techniques for use in concurrent and real-time systems; to confirm the utility of these techniques in a practical context; to determine and quantify the effectiveness of the techniques for enhancing software reliability; to measure the costs and overheads incurred as a consequence of adopting fault tolerance.

In order that the results of the project could be considered applicable and relevant to current practical systems it was decided to implement, for evaluation purposes, an application system of reasonable scale, constructed by professional programmers to normal commercial standards. The application selected was a medium-scale naval command and control system, engineered to be as realistic as possible, but incorporating software fault tolerance capabilities based on recovery blocks and conversations [1].

An experimental programme was designed which involved executing the application software with a simulated tactical environment using a large number of action scenarios. Two modes of execution were available, depending on whether the fault tolerance features were enabled or disabled. Data from these experiments was analysed to provide a number of quantitative assessments of the improvement in reliability arising from the use of fault tolerance. In fact, the results of this analysis suggest that software fault tolerance can prove very effective in coping with the consequences of faults in software.

This paper provides an overview of the experimental configuration, describes the programme of experiments, summarises the data obtained from the experiments, and presents the analysis of and results derived from this data. Information on costs is briefly summarised in the conclusions. A project report [12] provides full details of the experimental configuration and programme, and includes more details on costs. Another paper is being prepared which describes the software fault tolerance techniques developed for this project [13].

Experimental System Configuration

The experimental system employs three DEC computers configured as shown in figure 1. The command and control system runs on a PDP-11/45, and receives simulated sensor inputs from the simulator system running on an LSI-11/23. The simulator stores a representation of the tactical environment which is updated according to a predefined scenario file. Another PDP-11/45 provides file service facilities: scenarios for the simulator, and file storage to log the monitoring data generated by the command and control system.

All project software was written in the CORAL language, supported by a project developed MASCOT [14] executive. In particular the command and control system contained 8000 lines of CORAL source code structured into 14 concurrent activities, interacting as indicated in figure 2. To maximise the realism of this software, the Royal Navy participated in the high level design, programs were developed in accordance with MASCOT techniques, and documented to the MoD standard for miltary systems (JSP-188).

* CAP Scientific, London. ** RMCS, Shrivenham.

0731-3071/85/0000/0140$01.00 © 1985 IEEE

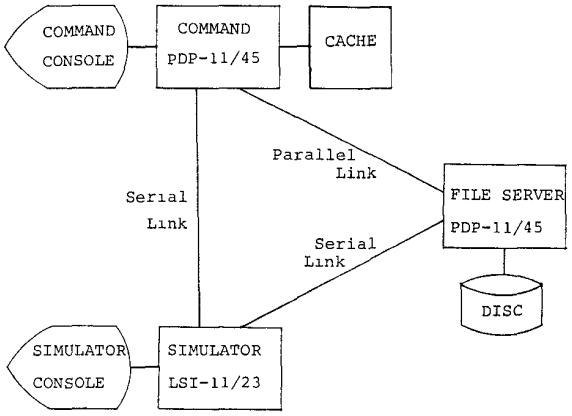

Figure 1 Hardware Configuration

The command and control system interacts with an operator and maintains a simulated radar display overlaid with tracking information. Using this information the operator can conduct an attack on a hostile submarine by means of a helicopter armed with a torpedo.

The provision of software fault tolerance in the command and control system was based on the use of recovery blocks [6] and 'dialogues' [13]. The dialogue notation provides a form of restricted conversation [1] which supports a static recovery structure for concurrent activities, while still permitting inter-process communication. Appropriate extensions and constraints were incorporated into the MASCOT executive to enable acceptance checks and alternative modules to be included in the command and control system. Backward error recovery [3] was provided by a prototype hardware recovery cache device [15] developed by a previous project.

Conduct of the Experimental Programme

The experimental programme consisted of running a number of typical scenarios on the command and control system. Each time an error was detected the operator would log the incident, and attempt to identify the fault which caused the error. The run would then continue, and the outcome of the incident (successful recovery or system failure) would be recorded. Monitoring routines within the system recorded data on all recovery and failure events, and this data was used to assist in classifying the events.

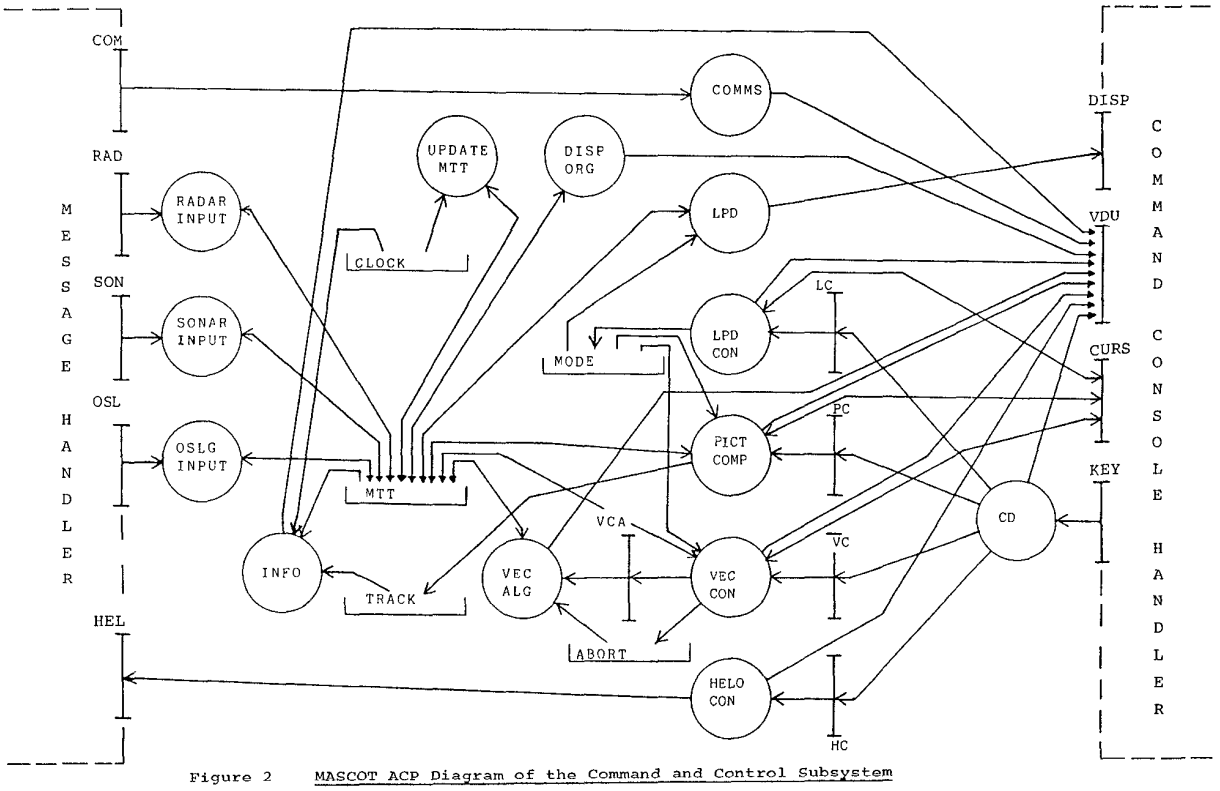

Figure 2 MASCOT ACP Diagram of the Command and Control Subsystem

The system was provided with a means of disabling the fault tolerance features. It was originally intended that each experiment would consist of a pair of runs, one conducted with fault tolerance enabled, the other with fault tolerance switched off. The intention was that the unrecoverable run would proceed along a similar path to the fault tolerant run until an event occurred. The unrecoverable run would then provide data on the consequences of that event in a non-fault tolerant system. A test exerciser sub-system (automatic operator) was constructed to run in conjunction with the command and control software in an attempt to provide a consistent operator reaction and thereby ensure repeatability. Experience soon showed, however, that the system did not provide the levels of repeatability required for such a method, and that two runs started in a similar manner were likely to follow quite different paths. The causes of this lack of repeatability are well understood, and centre around the unpredictability of the external interfaces to the command and control system. In particular, the communications protocol used to interface with the simulator (which uses a combination of checksums, timeouts and re-transmissions to ensure that no messages are lost or corrupted) is such that the ordering of the stream of messages from the simulated environment cannot be guaranteed to be the same for two runs conducted under similar conditions. Because of these problems fewer non-fault tolerant runs were performed, and these were used to provide information which could be used to predict the effects of an event on a non-fault tolerant system.

Experimental Programme Results

The results from the experimental programme are presented in two sections; the first section gives a summary of the events which occurred in the fault tolerant runs only, whereas the second section presents overall statistics on both sets of runs.

Summary of the Fault Tolerant Runs

An 'event' in a fault tolerant run is either an observed failure, or the detection by internal checks of a suspected error in the state of the system. Events were grouped into eight categories (enumerated in the tables below); it should be noted that in comparison with the non-fault tolerant version of the system, events in category 1 constituted an improvement in reliability, events in categories 2-5 produced no change in reliability, but events in categories 6 and 7 resulted in a deterioration in reliability.

The classification of events was performed at two levels of certainty. The first of these represents a very high level of confidence, being based on observation of the effects of the events on the system. Any event producing uncertain or unclear effects was placed in category 8. At the second level, the events in category 8 were allocated to the other categories according to the experimenters' judgement. The tables provide counts of the events in each category; figures in brackets record the second level counts which are less certain (although in practice the placings were made with a high degree of confidence).

Two cases are presented; the first covers all events whereas the second only includes the first event from each run. This distinction is made to factor out any events which might arise due to the inclusion of events which occurred **after** the non-fault tolerant system would have failed.

The total number of fault tolerant runs undertaken was **43.**

Summary of All Events in Fault Tolerant Runs

1. Events which produced recovery which averted failure	40	(40)
2. Events in which recovery occurred unnecessarily, but no failure resulted	4	(6)
3. Events in which a successful recovery took place, but the system failed	0	(0)
4. Events in which recovery was defective and the system failed	13	(14)
5. Events in which the system failed without recovery being attempted	0	(0)
6. Events in which a defective recovery caused the system to fail	3	(4)
7. Failures due to fault tolerance in which recovery was not attempted	1	(1)
8. Events for which the outcome is unclear	4	(0)

Total events: 65

Thus 40 events yielded an improvement, 17 events produced no change, and 4 events resulted in a deterioration of the reliability of the fault tolerant system in comparison with the non-fault tolerant version.

Summary of First Events in Fault Tolerant Runs

1. Events which produced recovery which averted failure	7	(7)
2. Events in which recovery occurred unnecessarily, but no failure resulted	4	(6)
3. Events in which a successful recovery took place, but the system failed	0	(0)
4. Events in which recovery was defective and the system failed	9	(10)
5. Events in which the system failed without recovery being attempted	0	(0)
6. Events in which a defective recovery caused the system to fail	2	(3)
7. Failures due to fault tolerance in which recovery was not attempted	1	(1)
8. Events in which the outcome is unclear	4	(0)

Total first events: 27

Comparative Data for Fault Tolerant and Non-Fault Tolerant Runs

Data in the previous section relates solely to runs of the fault tolerant version of the system, and assessment of the impact of fault tolerance on system reliability depends upon an analysis and categorisation of the events which took place. In this section, data is presented which enables a direct comparison to be made between the overall reliability of the two versions of the system. This may seem to provide a superior approach to comparative evaluation, but the reader is cautioned that due to a variety of factors (discussed below) the implications to be drawn from this data must be

stated less firmly than those based on the data of the preceding section.

The following table presents, for both versions of the command and control system, a summary of the entire experimental programme. It records the total number of experimental runs of each system, the total elapsed time during the execution of these runs, the total number of failures which occurred, and the number of runs which were completed without a failure of the command and control system (i.e. either completion of the scenario or premature termination due to a failure elsewhere).

	Fault Tolerant	**Non-Fault Tolerant**
Total runs	43	17
Total run time	50423 sec.	28057 sec.
Total failures	19	25
Failure-free runs	24	8

A number of points must be taken into consideration when analysing this data:

1. Firstly, the nature of a run should be considered. A run begins with a period of relative inactivity, during which little other than object and screen updating and object classification takes place, and during which very few events occur. This is followed by a phase during which the system supplies the operator with information enabling him to guide an armed helicopter to engage a target submarine. This phase is referred to as a "vectac" (vector and attack) and is a period of intense activity during which events are much more likely to occur. After the vectac, the system returns to relative inactivity until either a further vectac takes place or the run is stopped. During the experimental programme, in order to restrict runs to a manageable duration, and to ensure an adequate rate of occurrence of events, the periods of inactivity before and after a vectac were artificially curtailed. This was done by running the simulation in fast run' mode until shortly before the vectac was due to commence, then ending the run shortly after the vectac had completed (assuming that the system continued to run until this point). This curtailment has the effect that, since the system is likely to suffer few, if any, failures during periods of relative inactivity, any reliability measurements based on timing figures (for example MTBF) will appear far worse than they otherwise would. Thus, such figures might give a less favourable impression than figures for the proportion of events successfully recovered.

2. The lack of repeatability between runs (see section 3), the consequent lack of a one to one correspondence between fault tolerant and non-fault tolerant runs, and the divergence of the two systems when an event occurs, means that the implications of a direct comparison between the two systems are not as definitive as are the experimental results presented in the previous section.

3. The figures are heavily weighted by one particular run in which 16 of the total of 25 non-fault tolerant failures occurred. This run was in no way a 'freak'; all the failures were explained by known faults. However, the frequency of occurrence of such runs will clearly affect the overall system reliability. Unfortunately, there is insufficient data from the non-fault tolerant runs to deduce the probability that such a run will occur.

Analysis of Results

A number of different approaches can be adopted for estimating the increase in reliability which can be attributed to the provision of fault tolerance in the command and control system. Three approaches, characterising different aspects of reliability, are presented in the following sections. The first approach is based on estimating the "coverage" achieved by the fault tolerance techniques; that is, what proportion of potential failures are successfully averted thanks to software fault tolerance? The second approach provides a direct estimate of the mean time between failures for both versions of the system, while the third quantifies the proportion of missions successfully completed for the two versions of the system. A final section endeavours to estimate what results would have been obtained if the recovery mechanisms had been much more reliable.

Coverage Analysis

One measure of the effectiveness of the use of software fault tolerance in the demonstration system is a direct estimation of the proportion of failures which would have occurred in the non-fault tolerant version of the system which do not occur in the fault tolerant version of the system. To be a little more precise, given that a situation arises in which the non-fault tolerant system would fail, what is the probability that the fault tolerant system is able to continue operating without failing? The advantage of this approach is that the required probability can be estimated from the event count tables of the previous section, and this data is entirely discrete in nature. The probability is calculated as the ratio

$$\frac{\text{Number of failures averted}}{\text{Number of potential failures}}$$

A detailed calculation is presented for the data covering all events, since this is considered to provide the most satisfactory evaluation, addressing all events of the fault tolerant system on each experimental run.

Coverage of All Events

From the data for all events, the total number of events in fault tolerant runs was 65. Using the figures for high confidence of classification, 12 of these events are disregarded (initially), specifically 4 events in which the outcome is unclear (category 8), 4 events in which spurious recovery occurred (catgory 2) and 4 events in which a failure was caused by defective fault tolerance (categories 6 and 7). The remaining 53 events all constitute situations where the system would fail in the absence of fault tolerance. Of these 53 events, only 13 failures actually occurred in the fault tolerant system (categories 3, 4 and 5) since for 40 of the events recovery was invoked and failure was averted (category 1).

Thus the probability of success (coverage) should be estimated as 40/53, which is approximately 0.75. This is the maximum likelihood estimate. A Bayesian analysis using the Beta distribution indicates that the value estimated can be asserted to exceed 0.67 with 90% confidence.

These figures should be abated to take into account the four failures caused by fault tolerance. The

simplest approach regards these failures as "own goals" and subtracts them from the successes of category 1. An amended coverage estimate of 36/53 (i.e. 0.68) is then obtained.

Very similar results are obtained if the slightly less certain categorisation counts are used (those in brackets in the event count tables). The initial coverage ratio is then 40/54 (0.74), but five failures are now blamed on fault tolerance and so the adjusted coverage estimate falls to 0.65.

In summary, the above analysis shows that approximately 70% of potential software failures were prevented by the use of fault tolerance techniques in the command and control system software. This very encouraging result is perhaps the single most important finding of this research project.

Coverage up to First Event

The total number of first events in fault tolerant runs was 27. Using the high confidence data the initial estimate of coverage is determined as the ratio of category 1 divided by the sum of categories 1,3,4 and 5. This yields 7/16 giving a maximum likelihood estimate of 0.44, and a Bayesian 90% confidence point of 0.29. Allowing for failures induced by fault tolerance yields a coverage estimate of 0.25. The estimates derived from the less certain categorisation data are 0.41, and 0.18 allowing for induced failures.

In summary, this analysis shows that over 40% of potential software failures were prevented by means of fault tolerance even when only the first successful recovery in an experimental run is counted. The reduction in the estimate of coverage obtained from this section compared with that for all events suggests that after a first successful recovery had taken place further (possibly related) successful recoveries were likely to occur. Careful examination of the raw experimental data supports this contention. However, the count of failures actually caused by the provision of fault tolerance has a much greater impact on the results of this section. In fact, these failures can all be attributed (at least in part) to deficiencies in the implementation of the supporting recovery mechanisms. Results from which these effects have been eliminated are presented at the end of this section.

Failure Rate Analysis

Simple arithmetic applied to the table of timing and failure data yields the following results:

Failure rate for the fault tolerant system:	1.36 per hour
Failure rate for the non-fault tolerant system:	3.21 per hour
Ratio (fault tolerant / non-fault tolerant):	0.42

Making the standard, though often unjustified, assumption that the mean time between failures (MTBF) can be calculated as the reciprocal of the failure rate, yields the following alternative presentation of these results.

MTBF for fault tolerant system:	0.74 hours
MTBF for non-fault tolerant system:	0.31 hours
Ratio (fault tolerant / non-fault tolerant):	2.36

These results may be compared with those of the previous section by using the change in failure rate to provide an estimate for the coverage of failures by means of fault tolerance.

$$\text{Failure coverage: } \frac{3.21 - 1.36}{3.21} = 0.58$$

This value of 0.58 should be compared with the adjusted coverage estimate of 0.65 obtained above for all events. The agreement is reasonably close, and the measurements are mutually supportive. However, it should be remembered that the comparison between the fault tolerant and non-fault tolerant runs is by no means exact, because of the inability to precisely repeat any individual run.

Successful Missions

A further comparison between the two versions of the system may be made by examining the proportion of runs which were completed without a failure arising from the command and control system. Again, from the timing and failure data, it can be seen that the fault tolerant system is more reliable, though the improvement is much less marked.

Proportion of fault tolerant runs which completed without failing:	56%
Proportion of non-fault tolerant runs which completed without failing:	47%
Ratio (fault tolerant / non-fault tolerant)	1.19

Improved Recovery Mechanisms Projection

A substantial number of the failures which occurred during the fault tolerant runs were due to faults arising in the recovery mechanisms, either in the hardware recovery cache or the MASCOT recovery routines. This was perhaps to be expected in view of the prototype and problematic nature of the recovery cache, and because the MASCOT routines could not be comprehensively tested in advance due to resource limitations. In an operational system such failures could certainly be expected to occur very much less frequently, if at all, since the recovery hardware and software should be developed to stringent reliability standards. (The higher development cost of constructing reliable recovery mechanisms would be justified by their use in many projects as well as their crucial role.) Thus in this section an attempt is made to assess the outcome of the experimental programme assuming highly reliable recovery mechanisms. It can be argued that these figures give a more accurate estimate of the benefits which software fault tolerance can achieve than the actual experimental results presented in the preceeding sections.

The following figures were **not** explicitly achieved; they are obtained from the experimental results by eliminating 14 failures caused by defective recovery.

Failure coverage over all events:	0.91
Failure rate for the fault tolerant system:	0.36 per hour
MTBF for the fault tolerant system:	2.80 hours

MTBF ratio (fault tolerant / non-fault tolerant):	8.99
Comparative failure coverage	0.89
Proportion of fault tolerant runs successfully completed:	88%

In view of the significant improvement potential indicated by the above results it is planned to debug the recovery mechanisms in the light of this set of experiments, and then conduct a further phase of experimentation to see if these figures can be substantiated empirically.

Conclusion

The results of the previous section show clearly that for this application, in these experiments, the inclusion of software fault tolerance has produced a significant increase in reliability. Approximately 70% of software failures were eliminated, whereas the mean time between failures increased by about 135%. Furthermore, these improvements were achieved despite the use of essentially prototype hardware and software recovery mechanisms. Ignoring failures introduced by these mechanisms indicates that a failure coverage of about 90% could be achieved, with a nine-fold increase in MTBF.

Of course these gains were achieved at a cost, paid in capital costs to support fault tolerance, development costs to incorporate fault tolerance, and run time and storage overheads incurred by utilising fault tolerance.

The capital cost for supporting fault tolerance consisted of the cost of acquiring a hardware recovery device, for developing recovery software and incorporating this in the MASCOT operating system, and devising an interface by which dialogues and recovery blocks could interact with the operating system. The project expended approximately 1000 man-hours on these tasks, but the aim for the future would be that recovery facilities should be available on systems for critical applications on payment of a limited premium to the system manufacturer.

The supplementary development cost of incorporating fault tolerance in the command and control system was approximately 60%. This covered the provision of the acceptance tests and alternate modules used in recovery blocks and dialogues. The figure of 60% is probably rather high, reflecting the novelty of the techniques employed and their unoptimised utilisation in this particular application. Against the increased development cost must be offset any gains resulting in economies in testing the software.

Overheads in system operation were measured as: 33% extra code memory, 35% extra data memory and 40% additional run-time (though the system still had to meet its real-time constraints). The run-time overhead was incurred largely as a penalty for the synchronisation of processes for consistent recovery capability; data collection for state restoration purposes only contributed about 10% of the run-time overhead. By tuning the system to optimise its real-time response this overhead could be substantially reduced.

Thus, by means of software fault tolerance, a significant and worthwhile increase in software reliability was achieved at acceptable cost for a complex real-time system.

Acknowledgements

Over the two and a half year duration of this project we have received a great deal of assistance from many individuals, for which we are extremely grateful. We wish to express our gratitude explicitly to K. Heron, W. Lakin, B. Littlewood and B. Randell. The work was conducted with the financial support of the UK Science and Engineering Research Council and the Admiralty Surface Weapons Establishment of the MoD. We are grateful to these organisations and for the cooperation of the Royal Navy.

References

[1] B. Randell, **System Structure for Software Fault Tolerance**, IEEE Transactions on Software Engineering, SE-1(2), pp. 220-232, 1975.

[2] A. Avizienis, **Design Diversity - The Challenge of the Eighties**, Digest of FTCS-12, Santa Monica, pp. 44-45, 1982.

[3] T. Anderson and P. A. Lee, **Fault Tolerance: Principles and Practice**, Prentice/Hall 1981.

[4] T. Slivinski et al., **Study of Fault Tolerant Software Technology**, Report to NASA Langley Research Center, Mandex Inc., 1984.

[5] H. O. Welch, **Distributed Recovery Block Performance in a Real-Time Control Loop**, Proceedings of Real-Time Systems Symposium, Arlington, pp. 268-276, 1983.

[6] J. J. Horning et al., A **Program Structure for Error Detection and Recovery**, pp. 171-187 in Lecture Notes in Computer Science 16, Springer-Verlag, 1974.

[7] L. Chen and A. Avizienis, **N-Version Programming: A Fault- Tolerance Approach to Reliability of Software Operation**, Digest of FTCS-8, Toulouse, pp. 3-9, 1978.

[8] D. J. Martin, **Dissimilar Software in High Integrity Applications in Flight Controls**, AGARD Symp on Software for Avionics, The Hague, 1982

[9] O. B. von Linde, **Computers Can Now Perform Vital Functions Safely**, Railway Gazette International 135(11), pp. 1004-1006, 1979.

[10] R. K. Scott et al., **Modelling Fault-Tolerant Software Reliability**, Proc. of the 3rd Symposium on Reliability in Distributed Software and Database Systems, Clearwater Beach, 1983.

[11] J. Kelly and A. Avizienis, **A Specification Oriented Multi-version Software Experiment**, Digest of FTCS-13, Milan, pp. 120-126, 1983.

[12] T. Anderson et al., **Results and Conclusions from the Experimental Programme**, Fault Tolerance Project Report ref. 4844/DD.17/2, University of Newcastle upon Tyne, 1984.

[13] T. Anderson and M. R. Moulding, **Dialogues for Recovery Coordination in Concurrent Systems**, In preparation.

[14] Mascot Suppliers Association, **The Official Handbook of MASCOT**, Royal Signals and Radar Establishment, Malvern, 1980.

[15] P. A. Lee et al., **A Recovery Cache for the PDP-11**, IEEE Transactions on Computers, C-29(6), pp. 546-549, 1980.