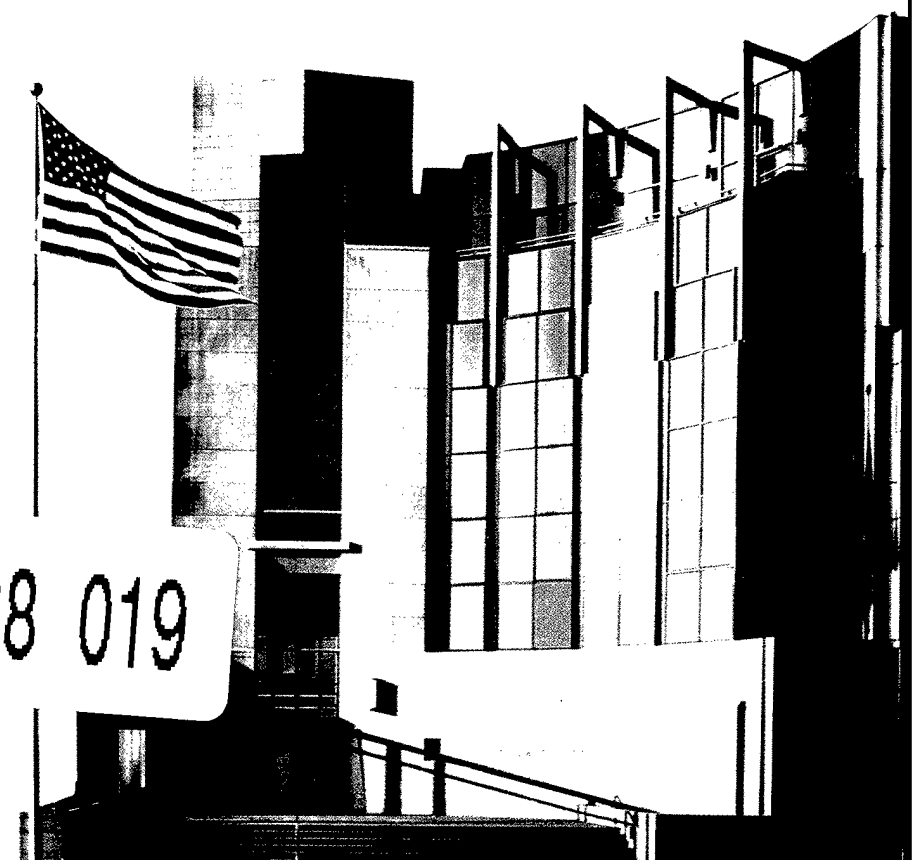
Carnegie Mellon University
Software Engineering Institute

A Perspective on the State of Research in Fault-Tolerant Systems

Charles B. Weinstock
David P. Gluch
June 1997

SR

Special Report
CMU/SEI-97-SR-008



Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of, "Don't ask, don't tell, don't pursue," excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Special Report
CMU/SEI-97-SR-008
June 1997

A Perspective on the State of Research in Fault-Tolerant Systems

Charles B. Weinstock
David P. Gluch

Dependable System Upgrade

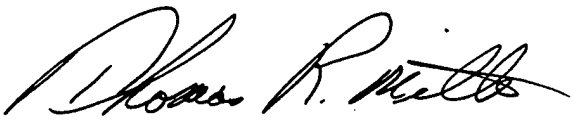
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through SAIC/ASSET: 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 / FAX: (304) 284-9001 / World Wide Web: http://www.saic.com/-contact.html / e-mail: webmaster@cpqm.saic.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

Table of Contents

Acknowledgments

The authors would like to thank Jacob Abraham of the University of Texas at Austin, and Peter Feiler, Mark Klein, Lui Sha, and Suzanne Couturiaux at the Software Engineering Institute for their review and helpful comments on this work.

A Perspective on the State of Research in Fault-Tolerant Systems

Abstract: As computers take on a greater role in society, their dependability is becoming increasingly important. Given software's critical role in computing systems, reliable software has emerged as crucial to achieving a dependable infrastructure. Using a system perspective that recognizes the prominence of software, we characterize the current state of fault-tolerance research as it contributes to the dependability of computer systems and we conjecture on future directions for this research area.

1 Introduction

We are becoming increasingly dependent on software-based systems, often without realizing it. While by no means can all such systems be classified as critical, software is turning up everywhere, from airplanes and automobiles to television sets and electric razors. Also, the percentage of software in these systems (relative to hardware) is increasing. According to Randell [Randell 95], the amount of software in consumer products is doubling every year (e.g., a top-of-the-line television now contains 500 kilobytes of software).

Consequences of failures in these systems can range from inconvenient (e.g., a television remote control that will not work) to catastrophic (e.g., software in a commercial aircraft that inadvertently prevents the pilot from recovering from an error.) The dependability of a television set may not be critical in the normal sense; however, for the manufacturer a television's low dependability can translate into lower sales and higher warranty repair costs. For the consumer an undependable television may result in the inability to obtain important information (e.g., school closings in a snow storm.)

Fault tolerance has long been used as a means of improving the dependability of computer systems. This report presents a perspective on research in fault tolerance as it relates to dependability in software-based systems and attempts to describe the current state of, and outline future directions for, this broad research field. While references are made to the commercial suppliers of fault-tolerant systems, this is not intended to be a survey of the capabilities of these systems.

The next section presents the system perspective that is used throughout this paper and discusses the problems involved in designing a system for dependability. Section 3 discusses the evolution of fault-tolerance research including the current state and future prospects. Finally, Section 4 presents a summary of the report.

2 Dimensions of the Problem Space

2.1 System Perspective

In the software engineering arena, a system is often equated with software, or perhaps with the combination of computer hardware and software. Here, we use the term *system* in its broader sense. As shown in Figure 2-1, a system is the entire set of components, both computer related and non-computer related, that provides a service to a user. For instance, an automobile is a system composed of many hundreds of components, some of which are likely to be computer subsystems running software.

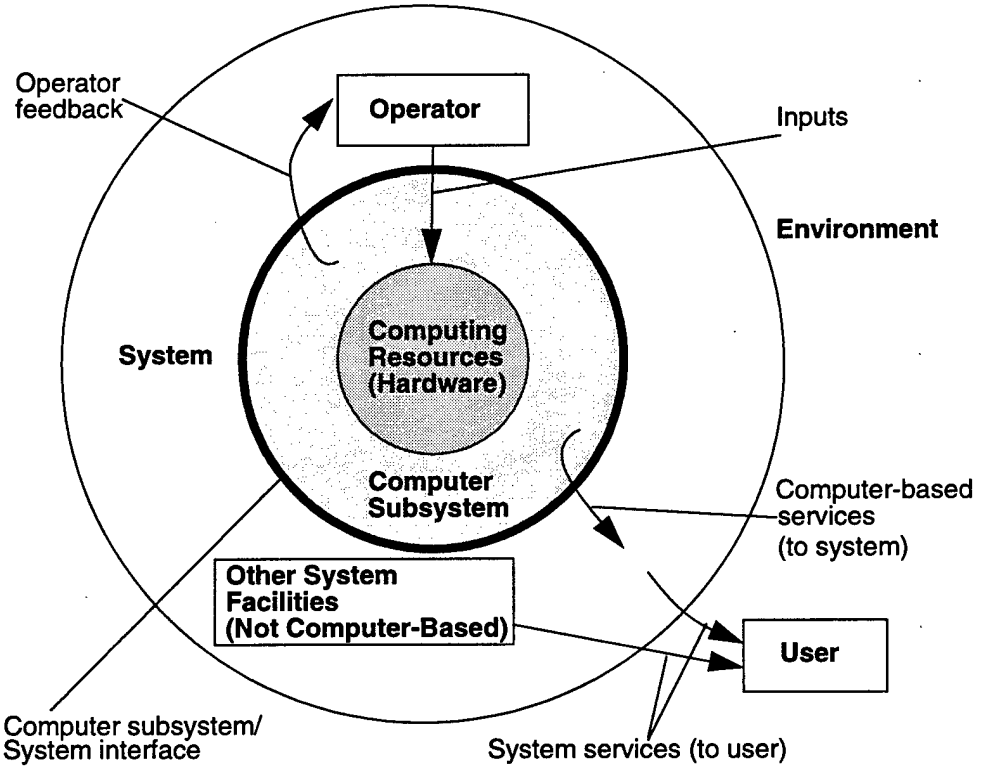


Figure 2-1 System Relationships

A system exists in an environment (e.g., a space probe in deep space), and has operators and users (possibly the same). The system provides feedback to the operator and services to the user. Operators are shown inside the system because operator procedures are usually a part of the system design; and many system functions, including fault recovery, may involve oper-

ator action. Not shown in the figure, but of equal importance, are the system's designers and maintainers.

Systems are developed to satisfy a set of requirements that meet a need. A requirement that is important in some systems is that they be dependable. A dependable system is one for which "reliance can justifiably be placed on the service it delivers [Laprie 92]." Fault tolerance is a means of achieving dependability.

There are three levels at which fault tolerance can be applied—hardware, software, and system (user interface). All three levels are susceptible to design, implementation, or maintenance errors—human mistakes that exist as faults in the hardware, code, or user interface and that are manifested in the behavior of the system. Hardware is unique among the three in that it is susceptible to "wear out" and damage. Traditional fault tolerance compensates for faults in computing resources (hardware). By managing extra hardware resources, the computer subsystem increases its ability to continue operation. Measures to ensure *hardware fault tolerance* include redundant communications, replicated processors, additional memory, and redundant power/energy supplies. Management of this redundancy often involves the use of software. Hardware fault tolerance was particularly important in the early days of computing, when the time between machine failures was measured in minutes.

A second level of fault tolerance recognizes that a fault-tolerant hardware platform does not, in itself, guarantee high availability to the system user. Faults can also arise from software components. These faults occur because the software was designed, implemented, or maintained incorrectly. *Software fault tolerance* (including mechanisms such as checkpoint/restart, recovery blocks, and multiple-version programs) is used to compensate for faults at this level.

A third level of fault tolerance realizes that software and hardware do not exist independently and that they execute in some environment. For instance, failures can arise as the result of actions by a user, whether a person or another computer system through the operational interface between the system and user. Measures taken at this level are usually application-specific and may be implemented in hardware or in software. At this level we have *system fault tolerance*.

The system view is recursive in nature. A subsystem may in itself consist of both hardware and software components. A simple example of this is a microprocessor whose instruction set is implemented in microcode. The microprocessor is subject to all of the usual kinds of hardware faults (e.g., single bit upsets), as well as all of the usual kinds of software faults (e.g., design errors in the microcode). The user of the microprocessor does not know or care which aspects of the subsystem are implemented in hardware or software. The microprocessor looks like a piece of hardware to most users. To the designer of the microcode, the microcode looks like software running on a specialized piece of hardware (the engine that executes the microcode.) The designer will have to deal with the hardware and software, which together make up the microprocessor, and be prepared to deal with the potential misuse of the microprocessor by the user.

2.2 Means to Achieve Dependability

The International Federation for Information Processing (IFIP) Working Group 10.4 has identified [Laprie 92] means to achieve improved dependability for a system. These can be categorized into fault-avoidance or fault-tolerance techniques. *Fault avoidance* prevents faults from occurring in the operational system and includes fault prevention, fault removal, and fault forecasting. *Fault tolerance* compensates for, and protects against, the impacts of faults during system operation.

Because the desired property of a system is improved dependability, software fault-tolerance research is not isolated from software fault avoidance issues. Understanding how faults occur and their nature is essential to defining strategies for tolerating the manifestation of faults in operational systems. Consequently, issues in fault-avoidance research are inseparable from considerations of fault-tolerance research.

2.3 Design for Dependability

When designing systems where dependability is required, it is important to deal with dependability issues from the start by including fault-tolerance mechanisms within the system design and employing fault-avoidance techniques as appropriate in the design process. Adding dependability after the fact can be both expensive and not as effective as designing it in from the start.

Depending on the nature of the system, tolerance of conventional faults or failures is not the only important consideration when designing dependable systems. Attention should also be paid to potential malicious "threats." As systems become more integrated, it is increasingly attractive to attack them, for teenage hackers as well as foreign terrorists. Attacks on the networked computer systems are becoming common, as evidenced by the increasing number of CERT® advisories appearing on the Internet. Much of our nation's infrastructure is becoming dependent on fragile, distributed software-dependent systems—the power grid, transportation (particularly air and rail), and our financial systems. These system are potential targets for intruder attacks.

2.4 Social and Economic Issues

The issues surrounding dependable systems are not just technical in nature. There are difficult social and economic barriers to be surmounted before dependability practices become standard in software-based systems. Although this report concentrates on technical areas, it is important to recognize the crucial role of these social and economic issues.

® registered in the U.S. Patent and Trademark Office.

Building dependable systems has a higher initial cost than building a system without paying special attention to dependability. Because of this, dependability has been a hard sell. Except in a limited set of critical applications with catastrophic failure modes (e.g., flight control, railroad signaling), there has been little perception or hard data to establish that the extra costs are worthwhile. This is changing. Whole industries are becoming dependent on software to operate, and the need for and cost effectiveness of higher levels of dependability are being recognized. Still, where the safety or economic benefit of dependability is not obvious, there is little demand for it.

There are cases where dependability (safety) drives the design (e.g., commercial aircraft). In others, regulations or economic incentives, such as severe penalties, may be needed to help push developers into designing for dependability.

Society demands dependability in other arenas—much more so than in software. Often, the demand for dependability is the result of some disaster or series of disasters that brings the dependability issue to the forefront. Leveson [Leveson 94] draws parallels between the proliferation of high-pressure steam engines in the early steam-era and computer software today. Boiler explosions and their aftermath led to the routine use of safety devices and to higher standards of workmanship and engineering. Similarly, it may require a major software-related disaster(s) to cause the public or management to routinely demand dependability guarantees on software-based systems.

Making systems dependable is difficult and requires systematic engineering, not just extensive user testing. The additional cost and/or time involved in achieving higher levels of dependability result in systems with a higher initial cost. Therefore if a consumer makes a purchase decision based only on price, ignoring the long-term costs associated with lower dependability, there is little incentive for the supplier to spend the additional resources on making the system dependable.

One reason that dependable systems cost more is that most fault-tolerant system architectures are proprietary. There is hope that increased demand will result in generic implementations which will bring the cost down. There is evidence that this is already happening. New fault-tolerant systems by Tandem [Tandem 96], Sequoia [Sequoia 96], and other manufacturers are based on UNIX and are much more open than their predecessors.

3 Evolution of Fault-Tolerance Research

Research in the field of computer system fault tolerance has evolved throughout the history of electronic computing. The unreliability of vacuum tubes and related discrete components of the earliest electronic computers necessitated conservative engineering and constant attention, if not extensive redundancy. For example, ENIAC included over 17,000 vacuum tubes and related components totalling more than 100,000 discrete parts. The key designers of the ENIAC realized that the success of the entire project rested on achieving component and system reliability [Goldstine 72].

As the field of computer design evolved, inherent component reliability and overall hardware reliability increased [Gray 91]. These improvements were due to the evolution of hardware technology (e.g., vacuum tubes to transistors to integrated circuits and higher levels of component integration), as well as the advancement of fault-avoidance and fault-tolerance techniques. Fault avoidance helped improve the quality of both the components and the systems, while fault tolerance found its greatest application in high-availability and continuous-performance systems, where computers began to take on increasingly critical roles in commercial and industrial endeavors (e.g., process control and information processing and storage).

An increase in overall complexity and capability accompanied the evolution of computer technology, with much of this increase implemented within software. Consequently, software became a larger and indispensable part of systems. With this growth, software began to emerge as a significant contributor to system unreliability [Gray 90].

While software reliability issues continue to play a critical role in achieving higher levels of dependability, the ubiquitous role of software has elevated human-computer interaction (HCI) issues into prominence. This situation is especially evident in applications where software-intensive systems have replaced traditional hardware-intensive systems as the primary technologies for information presentation and interaction (e.g., control panels replaced with graphical display units). Within these domains and in the operation of these systems, human error has been responsible for more downtime than generally recognized and, as a result, has had a significant effect on system performance and dependability [Maxion 96, Velpuri 95].

It is within the context of computer system evolution that fault-tolerance research has contributed substantially to improved system dependability. It is this evolution that has resulted in increasing research into the broader (non-hardware oriented) aspects of system dependability. Reliable software, user interfaces, and inter-application interfaces within software-intensive systems have emerged as important considerations in fault-tolerance research.

The broad focus has stimulated research into understanding the effect of software development practices on system reliability. In addition, researchers are investigating the characteristics of software architectures that will enable the attributes (specifically dependability) of software-intensive systems to be predicted.

3.1 Issues Facing Fault-Tolerance Researchers

As reflected in specialized conferences, sessions within general conferences, and journal publications, fault tolerance is a very active research field. Much of today's work is aimed at incremental improvement, maturing basic understanding in specific technology focus areas, and expanding the applicability of fault-tolerance technologies. There are no overriding technological approaches that dominate or help focus research, and the community is attacking pieces of the technology, problems, and solutions. This situation is in contrast to a few years ago where a central theme (i.e., n-version programming) typified much of the software fault-tolerance research and attracted a multitude of researchers.

3.1.1 Broad Range of Issues

This section includes a representative set of active research topics in fault tolerance. These range from fundamental fault-tolerance design and implementation issues (e.g., checkpoint restart, distributed algorithms) to fault-avoidance approaches (e.g., formal methods). The wide range of topics and the interweaving of fault-avoidance with classical fault-tolerance issues are indicative of the broad and complementary nature of these research areas.

Checkpoint Restart

Checkpoint restart strategies are backward recovery techniques for saving the state of a system to enable resuming operation from a well-defined state [Pradhan 96, Jalote 94]. This is classic fault-tolerance issue that is now being addressed in the context of increasingly complex and distributed systems. Much of the current research work involves issues relating to reliable high performance checkpointing [Plank 95] and distributed systems [Wang 95, Smith 95].

Distributed Algorithms

Unlike uniprocessor-based systems, distributed systems present a new set of challenges to achieving dependability. Clocks of the processors often must be synchronized, data must survive failures of individual processors, nodes must fail in controlled ways, and the communications between the processors must be reliable. Techniques such as interactive consistency, fail-stop processors, and reliable transport mechanisms have been developed to deal with these challenges. Many of the mechanisms are expensive to implement, and researchers continue to look for more efficient algorithms [Jalote 94].

Group Communications

In distributed systems high availability can be achieved by replicating state among groups of servers. As long as the state is consistent between the servers, the system is able to recover from the failure of one of them. This *replica consistency* is often achieved by using group membership protocols, which ensure that all running servers share a common view of the system configuration, and atomic broadcast, which ensures that each server is updated correctly. Work in this area is widespread, including the University of California at San Diego [Cristian 96] and Cornell University [Schneider 90].

Fault Tolerance in Human Computer Interaction (HCI)

Issues and perspectives in fault-tolerance research are expanding from internal software state issues to considerations of interfaces between systems and between humans and computers. This broadened perspective includes considerations of the environment and its effect on software and overall system dependability.

Questions such as, "How will this technology affect usability and safety?" and "How do human errors contribute to system downtime?" are being addressed [Neumann 95], [Gray 91]. These investigations involve both fault-tolerance and fault-avoidance issues in trying first to predict how errors will be introduced by human operators, under what conditions they will be introduced, what types of errors will be introduced, and how to tolerate these occurrences [Maxion 96, Velpuri 95, Ladkin 96].

Fault Injection

Fault injection is a technique for evaluating the dependability of a system. It's purpose is to test the ability of a system to detect and recover from faults. As a result of running fault-injection experiments, designers are able to determine the fault coverage of their system. Fault injection involves seeding the system with faults under controlled conditions and observing its behavior. Many tools have been developed to aid in fault injection including FIAT at Carnegie Mellon University [Barton 90], FERARRI at the University of Texas at Austin [Kanawati 95], and DEPEND at the University of Illinois [Ries 96].

Measurement and Interpretation

A significant difficulty in evaluating fault-tolerance and fault-avoidance mechanisms is the lack of real-world data. Without this data it is impossible to determine the efficacy of a method in a deployed system. Such data exist but are often proprietary. Some researchers have obtained access to such data and have been able to tune techniques and improve systems as a result. Notable examples of work in this area have taken place at the University of Illinois and Tandem Computers [Lee 95, Thakur 95], and at IBM's T. J. Watson Research Center [Chillarege 95].

Reliability Modeling

Reliability modeling is the modeling of faults and errors with the intention of predicting future behavior [Musa 90, Laprie 96, Farr 96]. Traditional hardware reliability models are empirically based and reflect physical characteristics of hardware failures, e.g., random failure models. In contrast, software and system failure models lack the physical data to guide reliability modeling and require reliance on usage data. Usage data is highly problematic both to collect and, because of the dependency of software reliability on its use, to generalize across systems.

Recent research in reliability models for predicting the future behavior of a system has focused on extending models, including complex and distributed systems, and addressing software reliability and its impact on overall system reliability. Summaries of these are available in a variety of publications [Pham 92, Pham 95, Lyu 96].

Formal Methods

The use of formal methods is a fault-avoidance technique which helps to achieve dependable systems by locating inconsistencies in their design or implementation. The application of formal methods to dependable system may involve detailed hand analysis, semi-automatic analysis (e.g., tools to help tabulate data), or the use of complex theorem provers. No matter which techniques are used, the use of formal methods can often find extremely subtle problems but scaling to large systems can be problematic. SRI International [Rushby 95], the National Aeronautics and Space Administration (NASA) [Bulter 95], and the Naval Research Laboratory (NRL) [Heitmeyer 96] are among the many organizations working in this area.

Object-Oriented Systems

Advances in the implementation of object-oriented systems and their expansion into critical and real-time applications has stimulated work attempting to integrate fault-tolerance and object-oriented technologies [Xu 95, Landis 95]. Specific efforts include addressing the critical fault-tolerance issues of partial failures and consistent ordering of events in distributed systems that are not adequately handled, for example, in Object Request Broker (ORB) or Common Object Request Broker Architecture (CORBA) technologies [Maffeis 95].

Innovative Applications

This research involves applying fault tolerance in new ways to solve problems that have not traditionally been associated with fault tolerance. It is based on abstracting the results of fault-tolerance research and mapping these to other problem spaces. Examples include the use of fault tolerance to enable dependable system upgrade [Sha 96] and the application of fault tolerance to information survivability and security problems [Randell 95, Wilken 95].

3.1.2 Technology Transition

An emerging perspective that is gaining greater emphasis is the challenge associated with transferring existing fault-tolerance technology into widespread practice.

Recently, the High Assurance Computing Workshop team (which was composed of 50 experts from academe, government, and industry), identified two primary problems associated with high-assurance (dependable) computing. Both problems involved the application of dependable computing to the development of practical systems. These problems were (1) lack of technology to support the application of dependability techniques and (2) lack of a unified framework for building systems that satisfy several critical properties (e.g., dependability and performance) simultaneously [McLean 95].

Evidence of efforts to transfer fault-tolerance techniques is seen in recent publications. For example, in the field of software fault tolerance two volumes edited by Michael Lyu [Lyu 95, Lyu 96] attempt to capture the state of software fault tolerance and software reliability. Similarly [Jalote 94, Pradhan 96, Siewiorek 92] capture broader design practices for fault-tolerant systems. These efforts are representative of the process of codifying the field toward integrating fault-tolerance techniques into more general engineering practice.

As with any transition challenge, the transition of fault-tolerance technologies must address not only technical but economic issues. This must include engineering studies and validated data on the impact (e.g., performance, cost) and the benefits (e.g., reduced downtime, lower maintenance costs) involved in engineering more dependable software-intensive systems.

3.1.3 Difficult Challenges

While one might conjecture that the field is mature, based on the emergence of the importance of transition, this is not a completely accurate description. Rather, while substantial success has been realized in the application of established techniques, the system and software fault-tolerance research communities still face complex problems that are too challenging for current technologies. As an example, published work has identified the difficulties in black box testing of software systems for high-reliability applications. In some cases, this work has described the infeasibility of proving ultra-high levels of software reliability, even with the use of advanced testing methods [Littlewood 91, Butler 93].

Similarly, challenges such as proving levels of independence among various software versions and establishing reliable (provable) error-detection mechanisms remain unanswered. For example, to determine if an error has occurred, a reference (a "correct" value) must be used. In hardware, the output of redundant components can be used as a reference. In this case, the challenge is the reliable implementation of the mechanism to compare outputs with the reference(s). In software the challenge is not only to implement the comparison but to establish a credible reference across the system's full operating environment, a reference that is itself sufficiently reliable.

Current fault-tolerance research areas are varied and there is no obvious central direction for future research activities. There is a sense in the community, as evidenced by informal discussions at conferences and workshops, of a need to develop more of a focus, or at least more clearly define the state of fault-tolerance research. The situation can be characterized as one that is opportune for a major breakthrough, perhaps a paradigm shift that would enable fundamental improvement in predictably achieving software and system dependability and safety.

3.2 Prospects for Fault-Tolerance Technologies

Building upon the assessment of the current state of fault-tolerance research, this section summarizes potential directions for technologies and research in fault tolerance. These are divided into those areas that are advancing the technology itself and those that are making fault-tolerance technology more widely available.

3.2.1 Advancing the Technology

The core of the activity in fault-tolerance research will center on advancing fundamental approaches within the discipline. These directions will likely involve four broad aspects: new breakthroughs, software, user issues, and proof of correctness.

New Breakthroughs

Redundancy and diversity of implementation have been cornerstones of system and software fault tolerance. But given the current situation, significant advancements in system fault tolerance may be realized only through a fundamentally new paradigm (e.g., a revolutionary fault-tolerance approach, the identification of methods that prove the degree of independence of various software versions, or dramatic advances in the application of formal methods or proof of correctness). Some of these approaches may involve post-implementation testing or modifications to software development practices. This work may very well profit from general research in understanding software engineering practices, software structure, and software attribute prediction.

Software Fault Tolerance

Software faults are idiosyncratic to a software implementation and represent logical or design errors made by human beings. While the use of software dictates how and when faults are manifested, software and software reliability embody the results of human efforts to interpret and understand the interactions of complex structure and behavior.

These logical or design errors in software mirror research problems found in software engineering generally. Much of the current and future research in software engineering seeks to establish patterns in the structure of software [Clements 95] and to better understand the practice of engineering software systems. In the future, fault tolerance research will find greater synergy with and benefit from the research in architectures, patterns, objects, and related technologies. These efforts may form the foundation for enabling greater predictability in the reliability and dependability attributes of software systems.

User Environment

Continuing issues related to the interplay of the user and the system will provide a substantial challenge for researchers. These issues will require more interdisciplinary work, especially between fault-tolerance and HCI researchers, and will require understanding the interrelationships between technology and its role within society.

Proof of Correctness

At least for critical parts of a system, such as error detection and recovery software, a concerted effort to find new directions in "proving" the correctness of software may be necessary. Some possible directions include the use of powerful abstractions to reduce the complexity and more "natural" specification languages which would be embraced by programmers. Such techniques are being applied successfully to complex hardware designs. (Intel, IBM, Motorola, AMD, HP, DEC, SUN, SGI, etc., all have formal verification groups; Intel, for example, has a large number of researchers working on the problem.[1])

---

1. The ideas expressed in this paragraph reflect a private communication between the authors and Jacob Abraham of the University of Texas at Austin.

3.2.2 Making the Technology Available

Much of the research in fault tolerance will continue to support making these techniques available to a broader community of practice and applying them to more domains. The exact direction of these efforts will likely be driven by the technologies that emerge as important to society (e.g., World Wide Web, mobile computing [Pradhan 96]). Work in this area will focus on the following three issues: repackaging and improving fault-tolerance technologies, transitioning the technology, and extending the technologies to other areas.

Repackage and Improve

Work will continue on new and optimized algorithms and enhanced implementations of fault-tolerance technologies. Much of this work will be aimed at structuring building blocks for use and reuse. For example, some of these advances may include higher quality, more robust algorithms for complex highly distributed systems; enhanced approximations toward ensuring near absolute fail-stop capabilities; and reuse of fault-tolerant software based upon architectural patterns.

Transition Mission

As fault-tolerance technology matures, increasing attention will likely turn to implementation and use in real systems. Investigations may include determining the practicality of techniques and whether these will scale to large real-world systems. These efforts will also need to consider the social and economic aspects of the technology.

Extend to Other Pressing Problem Areas

Efforts will likely continue in the application of fault-tolerance techniques to other problem spaces and in the integration of fault-tolerance techniques with related or supporting technologies. These directions will also foster a broad systems orientation in developing software-intensive systems. Current examples include the use of Simplex [Sha 96] for system upgrade, use of commercial off-the-shelf (COTS) components in dependable systems, integrating object-oriented approaches with fault tolerance in reflective programming [Xu 95], and the application of fault tolerance to information survivability problems [Randell 95].

4 Summary

This report presents a perspective on the directions of research in the field of fault tolerance for dependable computing systems. The field of fault tolerance encompasses hardware, software, system, and user issues. In the past, most research has centered on developing new techniques to achieve dependable systems and on the details of system design and performance. Increasingly, though, the issues of user errors and the effects of the environment of use are being recognized as critical to overall system dependability.

Reliable software will continue to be a key factor in achieving dependable system performance. Future research will need to address techniques for not only measuring but predicting the dependability of software-intensive systems. Within this area, opportunities will exist for synergy between fault tolerance and software engineering research.

While future progress will likely be incremental, the community appears poised for fundamentally new developments. As a context for these developments, and perhaps as precursors to them, important challenges exist. Some of these include

- software reliability prediction and measurement
- black-box testing
- user and usability issues
- repackaging and improvements
- reliable error detection
- proving the independence of software components
- extension to other problem domains
- transition of the technology to industry (real-world issues)
- information survivability

These problems are not just technical; there are also social and economic issues that have broad effect on the research and adoption of fault-tolerance technologies.

This is a living document that represents the current thinking of the authors. Readers are invited to help improve this document by sending comments to the authors at

dependable.software@sei.cmu.edu

References

[Barton 90] Barton, J.H.; Czeck, E.W.; Segall, Z.Z.; & Siewiorek, D.P. "Fault Injection Experiments using FIAT." *IEEE Transactions on Computers* 39, 4 (April 1990): 575-582.

[Butler 93] Butler, Ricky W. & Finelli, George B. "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software." *IEEE Transactions on Software Engineering* 19, 1 (January 1993): 3-12.

[Butler 95] Butler, Ricky W.; Caldwell, James L.; Carreno, Victor A.; Holloway, C. Michael; Miner, Paul S.; & Di Vito, Ben L. "NASA Langley's Research and Technology Transfer Program in Formal Methods." *Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS 95)*. Gaithersburg, MD, June 1995. New York, NY: IEEE, 1995.

[Chillarege 95] Chillarege, R; Biyani, S.; & Rosenthal, J., "Measurement of Failure Rate in Widely Distributed Software," 424-433. *25th International Symposium on Fault Tolerant Computing. Digest of Papers.* Pasadena, CA, June 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Clements 95] Clements, Paul C. *Coming Attractions in Software Architecture* (CMU/SEI-96-TR-008, ADA 309156). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[Cristian 96] Cristian, Flaviu. "Synchronous and Asynchronous Group Communication." *Communications of the ACM* 39, 4 (April 1996): 88-97.

[Farr 96] Farr, William. Ch. 3. "Software Reliability Modeling Survey." 71-117. *Software Reliability Engineering*, IEEE Society Press, Los Alamitos, California (1996).

[Goldstine 72] Goldstine, Herman H. *The Computer from Pascal to von Neuman.* Princeton, NJ: Princeton University Press, 1972.

[Gray 90] Gray, J. "A Census of Tandem System Availability Between 1985 and 1990." *IEEE Transactions on Reliability* 39, 4 (October 1990): 409-418.

[Gray 91] Gray, J. & Siewiorek, D.P. "High-Availability Computer Systems." *IEEE Computer* 24, 9 (September 1991): 39-48.

Heitmeyer 96] Heitmeyer, Constance & Mandrioli, Dino, eds., "Formal Methods for Real-Time Computing." *Trends in Software*, Volume 4. Chichester, UK: John Wiley, 1996.

[Jalote 94] Jalote, Pankaj. *Fault Tolerance in Distributed Systems.* Englewood Cliffs, NJ: Prentice Hall, 1994.

[Kanawati 95] Kanawati, G. A.; Kanawati, N. A.; & Abraham, J. A., "FERRARI: A Flexible Software-Based Fault and Error Injection System." *IEEE Transactions on Computers* 44, 2 (February 1995): 248-260.

[Ladkin 96] Ladkin, Peter. *Reasons and Causes* [online]. Available WWW: <URL:http://www.techfak.uni-bielefeld.de/~ladkin/Causes.html> (1996).

[Landis 95] Landis, S. & Stento, R. "CORBA with Fault Tolerance." *ObjectT Magazine* 5, 7 (November-December 1995): 62-66.

[Laprie 92] Laprie, J. C., ed. *Dependability: Basic Concepts and Terminology.* International Federation for Information Processing (IFIP) WG10.4 Dependable Computing and Fault Tolerance. New York, NY: Springer-Verlag, 1992.

[Laprie 96] Laprie, J. C. & Kanoun, Karama. Ch. 2, "Software Reliability and System reliability," 27-70. *Software Reliability Engineering*, Los Alamitos, California: IEEE Society Press, 1996.

[Lee 95] Lee, Inhwan & Iyer, R.K. "Software Dependability in the Tandem GUARDIAN System Source." *IEEE Transactions on Software Engineering* 21, 5 (1995): 455-467.

[Lyu 95] Lyu, Michael R. , ed. *Software Fault Tolerance.* Chichester, England: John Wiley & Sons, 1995.

[Leveson 94] Leveson, Nancy G. *Safeware: System Safety and Computers.* New York, NY: Addison Wesley, 1994.

[Littewood 91] Littlewood, Bev. Ch. 6, "Limits to Evaluation of Software dependability," 81-110. *Software Reliability and Metrics.* New York, NY: Elsevier Science Publishing Co., Inc., 1991.

[Lyu 95] Lyu, Michael R. , ed. *Software Fault Tolerance.* Chichester, England: John Wiley & Sons, 1995.

[Lyu 96] Lyu, Michael R., ed. *Software Reliability Engineering.* Los Alamitos, California: IEEE Computer Society Press, 1996.

[Maffeis 95] Maffeis, Silvano. "Adding Group Communication and Fault-Tolerance to CORBA." *Proceedings of the USENIX Conference on Object-Oriented Technologies.* Monterey, CA, June 1995. Berkeley, CA: USENIX Association, 1995.

[Maxion 96] Maxion, Roy A. & Syme, Philip A. Mitigating Operator-Induced Unavailability by Matching Imprecise Queries," 240-249. *26th International Symposium on Fault Tolerant Computing, Digest of Papers.* Sendai, Japan, June 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[McLean 95] McLean, John & Heitmeyer, Constance. *High Assurance Computer Systems: A Research Agenda (*Workshop Report, February 21-23, 1995). Washington, DC: Naval Research Laboratory, 1995.

[Musa 90] Musa, John D.; Iannino, Anthony; & Okumoto, Kazuhira. *Software Reliability: Measurement, Prediction, Application.* New York, NY: McGraw-Hill Publishing Company, 1990.

[Neumann 95] Neumann, Peter G. *Computer Related Risks.* New York, NY: Addison-Wesley/ACM Press, 1995.

[Pham 92] Pham, Hoang, ed. *Software Systems: Techniques and Applications.* Los Alamitos, CA: IEEE Computer Society Press, 1992.

[Pham 95] Pham, Hoang. *Software Reliability and Testing.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Plank 95] Plank, James S.; Kim, Youngbae; & Dongarra, Jack J. "Algorithm-Based Diskless Checkpoints for Fault tolerant Matrix Operations," 351-360. *25th International Symposium on Fault Tolerant Computing, Digest of Papers.* Pasadena, CA, June 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Pradhan 96] Pradhan, Dhiraj K. *Fault-Tolerant Computer System Design.* Upper Saddle River, NJ: Prentice-Hall, Inc., 1996.

[Randell 95] Randell, B. "Fault Tolerance and Security," 389-391. *Proceedings of Fourth IFIP Working Conference on Dependable Computing for Critical Applications.* San Diego, CA, January 4-6, 1994. New York: Springer -Verlag, 1995.

[Ries 96] Ries, G.; Kalbarczyk, Z.; Kraljevic, T.; Hsueh, M.-C.; & Iyer, R. K. "DEPEND: A Simulation Environment for System Dependability Modeling and Evaluation." *Proceedings of the Second IEEE International Computer Performance and Dependability Symposium (IPDS '96),* September, 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Rushby 95] Rushby, John. *Formal Methods and their Role in the Certification of Critical Systems (*SRI-CSL-95-1). Menlo Park, CA: SRI International, 1995.

[Scheider 90] Schneider, Fred B. "Implementing Fault-Tolerant Services Using the State Machine Approach: a Tutorial." *Computing Surveys* 22, 4 (December 1990): 299-319.

[Sequoia 96] Sequoia Enterprise Systems [online]. Available WWW: <URL: http://seqweb.sequoia.com/ser400.html>.

[Sha 96] Sha, Lui R.; Rajkumar, Ragunathan; & Gagliardi, Michael. "Evolving Dependable Systems," 335-346. *Proceedings of 1996 IEEE Aerospace Applications Conference on Reliability and Quality of Design, Part 1.* Aspen, CO, Feb, 1996. New York: IEEE, 1996.

[Siewiorek 92] Siewiorek, Daniel P. & Swarz, Robert S. *Reliable Computer Sys-Design and Evaluation.* Burlington, MA: Digital Press, Inc., 1992.

[Smith 95] Smith, Sean W.; Johnson, David B.; & Tygar, J. D. "Completely Asynchronous Optimistic Recovery with Minimal Rollbacks," 361-370. *25th International Symposium on Fault Tolerant Computing, Digest of Papers.* Pasadena, CA, June 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Tandem 96] Tandem Computer Home Page [online]. Available WWW: <URL: http://www.tandem.com/> 1996.

[Thakur 95] Thakur, A.; Iyer, R. K.; Young, L; & Lee, I. "Analysis of Failures in the Tandem NonStop-UX Operating System." 40-49. *Proc. International Symposium on Software Reliability Engineering*, Toulouse, France, October 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Velpuri 95] Velpuri, Rama. *Oracle Backup Handbook.* Berkeley, CA: Oracle Press/Osborne McGraw-Hill, 1995.

[Wang 95] Wang, Yi-Min; Chung, Pi-Yu; Lin, In-Jen; & Fuchs, W.K. "Checkpoint Space Reclamation for Uncoordinated Checkpointing in Message-Passing Systems." *IEEE Transactions on Parallel and Distributed Systems* 6, 5 (May 1995): 546-54.

[Wilken 95] Wilken, K.D. "Common techniques in Fault Tolerance and Security (and Performance!)," 393-395. *Proceedings of Fourth IFIP Working Conference on Dependable Computing for Critical Applications.* San Diego, CA, January 4-6, 1994. New York: Springer - Verlag, 1995.

[Xu 95] Xu, Jie; Randell, Brian; Romanovsky, Alexander; Rubira, Cecilia M. F.; Stroud, Robert; & Wu, Zhixue. "Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery," 499-508. *25th International Symposium on Fault Tolerant Computing. Digest of Papers.* Pasadena, CA, June 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-97-SR-008	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (city, state, and zip code) HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office 8b. OFFICE SYMBOL (if applicable) ESC/AXS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-95-C-0003
8c. ADDRESS (city, state, and zip code)) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO: 63756E PROJECT NO.: N/A TASK NO: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) A Perspective on the State of Research in Fault-Tolerant Systems	
12. PERSONAL AUTHOR(S) Charles B. Weinstock, David P. Gluch	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM TO	14. DATE OF REPORT (year, month, day) June 1997 15. PAGE COUNT 21
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (continue on reverse of necessary and identify by block number) computing systems, fault-tolerance research, fault-tolerant systems, software reliability

19. ABSTRACT (continue on reverse if necessary and identify by block number)

As computers take on a greater role in society, their dependability is becoming increasingly important. Given software's critical role in computing systems, reliable software has emerged as crucial to achieving a dependable infrastructure. Using a system perspective that recognizes the prominence of software, we characterize the current state of fault-tolerance research as it contributes to the dependability of computer systems and we conjecture on future directions for this research area.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution
22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF	22b. TELEPHONE NUMBER (include area code) (412) 268-7631 22c. OFFICE SYMBOL ESC/AXS (SEI) JPO)