FROM DEFECTS TO FAILURES: A VIEW OF DEPENDABLE COMPUTING

Behrooz Parhami

Dept. of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106, USA

Abstract: A unified framework and terminology for the study of computer system dependability is presented. Impairments to dependability are viewed from six abstraction levels. It is argued that all of these levels are useful, in the sense that proven dependability procurement techniques can be applied at each level, and that it is beneficial to have distinct, precisely defined terminology for describing impairments to and procurement strategies for computer system dependability at these levels. The six levels in the proposed framework are:

1. Defect level or component level, dealing with deviant atomic parts.
2. Fault level or logic level, dealing with deviant logic values or path selections.
3. Error level or information level, dealing with deviant internal states.
4. Malfunction level or system level, dealing with deviant functional behavior.
5. Degradation level or service level, dealing with deviant performance.
6. Failure level or result level, dealing with deviant outputs or actions.

Briefly, a hardware or software component may be *defective* (perfect hardware may also become defective due to wear and aging). Certain system states expose the ***defect***, resulting in the development of a logic-level ***fault***. Information flowing within a ***faulty*** system may become contaminated, leading to the presence of an ***error***. An ***erroneous*** system state may result in a subsystem ***malfunction***. Automatic or manually controlled reconfiguration can isolate or bypass the *malfunctioning* subsystem but may lead to a ***degraded*** performance or service. Serious performance ***degradation*** may lead to a result-level system ***failure*** when untrustworthy or untimely results are produced. Finally, a *failed* computer system can have adverse effects on the larger societal or corporate system into which it is incorporated.

1. INTRODUCTION

The field of dependable computing is an outgrowth of "fault-tolerant computing," a term which was introduced in the mid 1960s [PIER65], [AVIZ67], following more than a decade of concern with reliability issues [EJCC53], [MOOR56], [VONN56]. Tolerance of faults is actually an age-old design technique. For example, the engineer's "safety factor" is a means of tolerating faults of unknown origins and extents. Avizienis [AVIZ82a] quotes a 1.5-century old paper which advocates, in connection with Babbage's work, the use of different computers and algorithms for the purpose of checking computation errors. Forster [FORS28] fancies a civilization controlled by self-repairing machines which eventually self-destructs because the possibility of faults in the "Mending Apparatus" itself was not taken into account by the system's designers. Following the initial emphasis on fault tolerance through fault masking and self-repair, it was realized that "tolerance of faults" is but one way of achieving high reliability. This broadening of scope was reflected in the term "reliable computing," a variant of which had been used as far back as 1963 for complementing reliable communication [WINO63].

To avoid confusion between "reliability" as a precisely defined statistical measure and "reliability" as a qualitative attribute of systems and computations, use of "dependability" was proposed to convey the second meaning [LAPR82], resulting in the name "dependable computing." Thus, dependable computing deals with impairments to dependability (defects, faults, errors, malfunctions, degradations, failures, and crashes), means for coping with them (fault avoidance, fault tolerance, design validation, failure confinement, etc.), and measures of success in designing dependable computer systems (reliability, availability, performability, safety, etc.).

As computers are used for more demanding and critical applications by an increasing number of minimally trained users, the dependability of computer hardware and software becomes even more important. Highly dependable systems have been in widespread use for more than a quarter of a century [DOWN64] and have been marketed commercially for over a decade [KATZ77]. It is clear that with the emphasis on intelligent computers for the future, computer system dependability should become an integral part of the design process for future generation systems.

2. WHAT IS DEPENDABILITY?

We start by reviewing several definitions of (computer) system dependability. In one of the early proposals, dependability is defined briefly as [HOSF60]:

"the probability that a system will be able to operate when needed."

This simplistic definition, which subsumes both of the well-known notions of reliability and availability, is only valid for systems with a single catastrophic failure mode; i.e., systems that are either completely operational or totally failed. The problem lies in the phrase "be able to operate." What we are actually interested in is "task accomplishment" rather than "system operation." Laprie's definition is more suitable in this respect [LAPR82]:

"... dependability [is defined] as the ability of a system to accomplish the tasks (or equivalently, to provide the service[s]) which are expected from it."

This definition does have its own weaknesses. For one thing, the common notion of "specified behavior" has been replaced by "expected behavior" so that possible specification slips are accomodated in addition to the usual design and implementation inadequacies. However, if our expectations are realistic and precise, they might be considered as simply another form of system specification (possibly a higher-level one). However, if we expect too much from a system, then this definition is an invitation to blame our misguided expectations on the system's undependability. Carter provides a more useful definition [CART82]:

"... dependability may be defined as the trustworthiness and continuity of computer system service such that reliance can justifiably be placed on this service."

This definition has two positive aspects: It takes the time element into account explicitly ("continuity") and stresses the need for dependability validation ("justifiably"). Laprie's version of this definition [LAPR85] can be considered a step backwards in that it substitutes "quality" for "trustworthiness and continuity." The notions of "quality" and "quality assurance" are well-known in many engineering disciplines and their use in connection with computing (e.g., [BEIZ84], [DUNN82]) is a welcome trend. However, precision need not be sacrificed for compatibility.

To present a suitable definition of dependable computing, we need to examine the various aspects of undependability. From a user's viewpoint, undependability shows up in the form of *late, incomplete, inaccurate,* or *incorrect* results or actions [PARH78]. The two notions of *trustworthiness* (correctness, accuracy) and *timeliness* can be abstracted from the above; completeness need not be dealt with separately since any missing result or action can be considered to be (infinitely) late. We also note that dependability should not be considered an intrinsic property of a computer system. Rather, it should be defined with respect to particular (classes of) computations and/or interactions. Dependability cannot be discussed in absolute terms, just as "fault tolerance" would be meaningless without an explicit or implicit specification of the class of "tolerable faults." It is quite possible to design a computer system that trades off performance for dependability. One can envisage different system configurations and resource management procedures for various levels of dependability specified by the users. Thus, a system that is physically quite unreliable might become dependable by virtue of algorithmic dependability procurement mechanisms. Thus we are led to the following:

Dependability of a computer system may be defined as justifiable confidence that it will perform specified actions or deliver specified results in a trustworthy and timely manner.

Note that the above definition does not preclude the possibility of having various levels of importance for different (classes of) user interactions or varying levels of criticality for situations in which the computer is required to react. Such variations simply correspond to different levels of *confidence* and *dependability.* Our definition retains the positive elements of previous definitions, while presenting a result-level view of the time dimension by replacing the notion of "service continuity" by "timeliness" of actions or results.

Dependability (or *trustworthiness,* as some software researchers prefer to use) is actually a generic term that covers several system qualities such as *integrity, robustness, resilience, maintainability,* and the like. Various aspects of dependability are quantified by measures such as *reliability, availability, performability, testability,* and *safety.* Measures of dependability are discussed in Section 6 of this paper.

3. IMPAIRMENTS TO DEPENDABILITY

Impairment to dependability are variously described as *hazards, defects, faults, errors, malfunctions, failures,* and *crashes.* There are no universally agreed upon definitions for these terms, causing different and sometimes conflicting usages within the field of dependable computing. Although some research groups and authors have tried to present precise definitions for the terms and be consistent in their usage, progress towards accepting a standard terminology has been quite slow. There are two major proposals on how to view and describe impairments to dependability.

Members of the Newcastle Reliability Project [SHRI85], led by Professor Brian Randell, have advocated a hierarchic view [ANDE82]: A (computer) system is a set of *components* (themselves systems) which interact according to a *design* (another system). The recursion stops when we arrive at atomic systems whose internal structures are of no interest at the particular level of detail with which we are concerned. System *failure* is defined as deviation of its behavior from that predicted (required) by the system's authoratative specification. Such a behavioral deviation results from erroneous system state. An *error* is a part of an erroneous state which constitutes a difference from a valid state. The cause of the invalid state transition which first establishes an erroneous state, is a *fault* in a system component or in the system's design. Similarly, the component's or design's failure can be attributed to an erroneous state within the corresponding (sub)system resulting from a component or design fault, and so on. Therefore, at each level of the hierarchy, *"the manifestation of a fault will produce errors in the state of the system, which could lead to a failure."*

With this model, "failure" and "fault" are simply different views of the same phenomenon (the failure-fault model of Kopetz [KOPE82] is similar in its hierarchic view of the cause-effect relationships). This is quite elegant and enlightening but introduces some problems by the need for continual establishment of frames of reference when talking about the causes (faults) and effects (failures) of deviant system behavior at various levels of abstraction. While it is true that a computer system may be viewed at many different levels of abstraction, it is also true that some of these levels have proved more useful in practice. Avizienis [AVIZ82] takes four of these levels and proposes the use of distinct terminology for impairments to dependability (*"undesired events"*, in his words) at each of these levels. His proposal can be summarized in the following cause-effect diagram:

Abstraction level:	Physical	Logical	Informational	External
Undesired Event:	Failure ----------->	Fault -------------->	Error -------------->	Crash

In what follows, we extend and refine Avizienis's model. There are some problems with the above choices of names for undesired events. The term "failure" has traditionally been used both at the lowest and the highest levels of abstraction. Thus, we have *failure rate, failure mode,* and *failure mechanism* used by electrical engineers and device physicists alongside *system failure, fail-soft operation,* and *fail-safe system* coming from computer system designers. To comply with the philosophy of distinct naming for different levels, Avizienis retains "failure" at the physical level and uses "crash" for the other end. However, this latter term is unsuitable. Inaccuracies or delays, beyond what is expected according to system specifications, can hardly be considered "crashes" in the ordinary sense of the term. Besides, this term emphasizes system operation rather than task accomplishment and is thus unsuitable for fail-soft computer systems which, like airplanes, *fail* much more often than they *crash*!

Another problem is that there are actually three external views of a computer system. The maintainer's external view consists of a set of interacting subsystems that must be monitored for detecting possible malfunctions in order to reconfigure the system or, alternatively, to guard against hazardous consequences (such as total system loss or crash). The operator's external view consists of a black box capable of providing certain services and is more abstract than the maintainer's system-level view. Finally, the end user's external view is shaped by the system's reaction to particular situations or requests. Thus, we are led to a six-level view of the impairments to dependability:

Abstraction level:	Component	Logic	Information	System	Service	Result
Impairment:	Defect ---------->	Fault ---------->	Error ---------->	Malfunction --->	Degradation -->	Failure

Taking into account the fact that a non-atomic component is itself a system at a lower level, usage of the term "failure" in *failure rate, failure mode,* and *failure mechanism* can be explained by noting that the component is the end product (system) from the component designer's point of view. We can be consistent by always associating the term "failure" with the highest and the term "defect" with the lowest level of abstraction. Thus, the component designer's "failed system" is the system architect's "defective component."

4. SOME ANALOGIES

Let us use some familiar every-day phenomena to accentuate the chain of cause-effect relationships in our model. An automobile brake system with a weak joint in the brake fluid piping is *defective*. The weakness may have been caused by a design flaw (possibly resulting in a mass recall before the following undesirable effects), some road hazard, a chemical reaction, etc. If the weak joint breaks down, the brake system becomes *faulty*. A careful (off-line) inspection of the automobile can reveal the fault. However, the driver does not automatically notice the fault (on-line) while driving. The brake system state becomes *erroneous* when the brake fluid level drops dangerously low as a result of leakage through the broken joint. Again, the error is not automatically noticed by the driver, unless a brake fluid indicator light is present and properly functioning. At this point, the brake system is not yet *malfunctioning*, since it is not being used. The eventual malfunction is a result of inadequate brake fluid pressure upon the application of the brake pedal. Assuming that the automobile does not have a brake fluid indicator light, the driver's first realization that something is wrong comes from noticing the *degraded* performance of the brake system (higher force needed or lower resulting deceleration). If this degraded performance is insufficient for slowing down or stopping the vehicle when the need arises, the brake system has *failed* to deliver the expected result or to act properly.

As a second example, we consider a small organization. *Defects* in the organization's staff promotions policies may cause improper promotions, viewed as *faults*. The resulting ineptitudes and dissatisfactions may be considered *errors* in the organization's state. The organization's departments probably start to *malfunction* as a result of the errors, in turn causing an overall *degradation* of performance. The end result may be the organization's *failure* to achieve its goals. Obvious parallels exist between the various organizational procedures and dependable computing terms such as *fault testing* (staff reviews), *fault tolerance* (friendly relations and teamwork), *error correction* (personnel transfers), and *self-repair* (on-the-job training and alternate rewards).

Figure 1 provides a particularly interesting analogy, although it is less accurate than the preceding analogies in that it treats all levels uniformly. The six concentric water reservoirs correspond to the six levels of our model. Pouring water from above corresponds to defects, faults, errors, and other undesired events, depending on the layer(s) being affected. These undesired events can be avoided by controlling the flow of water through valves or tolerated by the provision of drains of acceptable capacities for the reservoirs. If water ever gets to the outermost reservoir, the system has failed. This may happen, for example, as a result of a broken valve at some layer combined with inadequate drainage at the same and all outer layers. The heights of the walls between adjacent reservoirs correspond to the natural inter-level latencies in our multi-level model. The outside world is represented by the area surrounding the reservoirs. Water overflowing from the outermost reservoir into the surrounding area corresponds to a computer failure affecting the larger corporate or societal system.

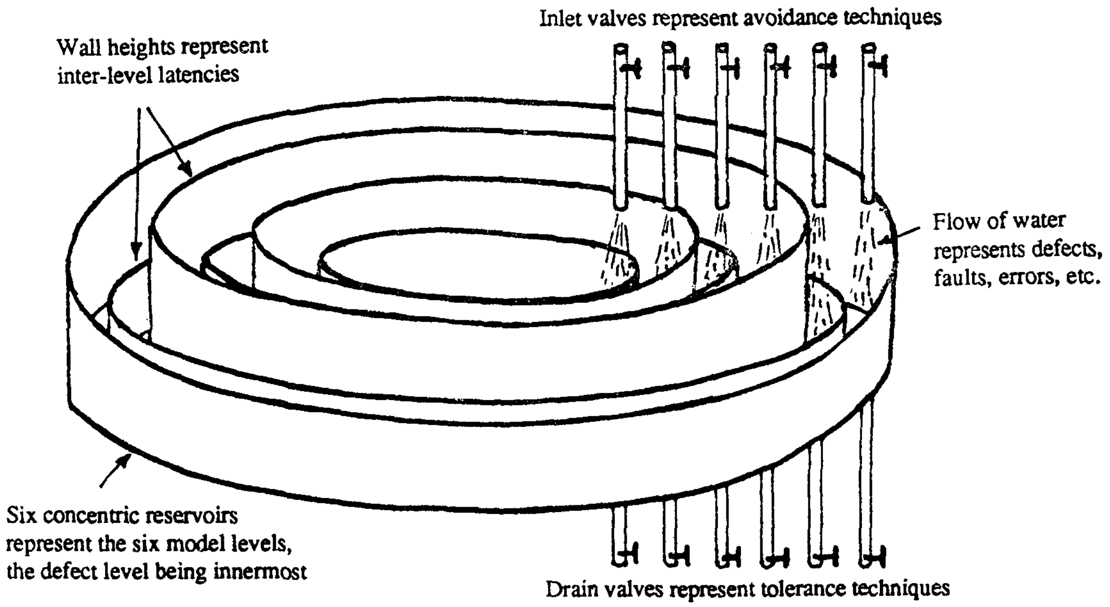
Inlet valves represent avoidance techniques
Wall heights represent inter-level latencies
Flow of water represents defects, faults, errors, etc.
Six concentric reservoirs represent the six model levels, the defect level being innermost
Drain valves represent tolerance techniques

Figure 1. A Simple Analogy for Our Multi-Level Model of Dependable Computing.

5. THE FIELD OF DEPENDABLE COMPUTING

The field of dependable computing deals with the procurement, forecasting, and validation of computer system dependability. As discussed in Section 3, impairments to dependability can be viewed from six levels. Thus, the subfields of dependable computing can be thought of as dealing with some aspects of one or more of these levels. Specifically, we take the view that a system can be in one of seven states: Ideal, Defective, Faulty, Erroneous, Malfunctioning, Degraded, or Failed. Note that these states have nothing to do with whether or not the system is "working." A system may be "working" even in the failed state; the fact that it is failed simply means that it isn't delivering what is expected of it. Upon the completion of its design and implementation, a system may end up in any one of the seven states, depending on the appropriateness and thoroughness of validation efforts. Once in the initial state, the system moves from one state to another as a result of *deviations* and *remedies*. Deviations are events that take the system to a lower (less desirable) state, while remedies are techniques or measures that enable a system to make the transition to a higher state.

Figure 2 shows the various system states and state transitions. Each state can be entered initially (through the sideways transition), from above as a result of a deviaiton (undesired event), or from below as a result of a remedy. Associated with each transition, we have five tags labeled as:

- *c* : (Natural) *cause* of the transition.
- *i* : (Natural) *impediment* to the transition.
- *f* : Techniques for *facilitating* the transition.
- *a* : Techniques for *avoiding* the transition.
- *m* : Tools for *modeling* the transition.

Recognized subfields of dependable computing deal with one or more of the above tags and have been shown in upper case in Figure 2. Other labels do not correspond to recognized subfields but describe various concepts and techniques of the field. In fact, the transition tags of Figure 2 can be used for accurate indexing of reference sources in dependable computing. For example, a paper may be described as dealing with *a*[→ faulty] and *i*[faulty→ erroneous].

Detailed discussions of the non-ideal states and their related transitions appear in an extended survey by this author that has been submitted for publication [PARH88]. Here, we present some general observations on the various system states. First, we note that the observability of the system state (ease of external recognition that the system is in a particular state) increases as we move downward in Figure 2. For example, the inference that a system is "ideal" can only be made through formal proof techniques; a proposition which is currently impossible for practical computer systems in view of their complexity. At the other extreme, a failed system can usually be recognized with little or no effort. As examples of intermediate states, the "faulty" state is recognizable by extensive off-line testing, while the "malfunctioning" state is observable by on-line monitoring with moderate effort. It is therefore common practice to force a system into a *lower* state (e.g., from "defective" to "faulty" by means of *burn-in* or *torture testing* of components) in order to deduce its initial state.

Defects are avoided through component *screening*, systematically removed by *preventive maintenance*, and tolerated through *component-level redundancy* techniques. Faults are avoided and removed through *fault testing* and tolerated by means of *logic-level redundancy* techniques such as *self-checking* design, logic circuit *duplication*, and signal-level *voting*. Note that the restricted use of the term "fault tolerance" in our model and terminology is in contrast to the traditional use of the term to denote the entire field of dependable computing [AVIZ78], [PRAD86], [NELS87]. Continuing with the states in our model, we note that errors are avoided through system*validation*, removed by *error detection* and the associated *recovery* mechanisms, and tolerated through the use of *error-correcting codes*. Malfunctions are identified by *malfunction diagnosis* (currently known as system-level fault diagnosis) techniques and are tolerated by *system-level recovery* and *resource redundancy*. *Design diversity* [AVIZ82a] incorporated in a *recovery block* architecture [RAND78] or in a replaication-and-voting arrangement is a malfunction tolerance technique that extends the protection provided against random independent malfunctions to related or common-cause malfunctions and also to design slips. Degradation detection and management is essentially a software task that is normally built into the operating system with extensive hardware support. Degradation tolerance, on the other hand, is both a system function and a property that can be incorporated into applications. Finally, failures are detected by external monitoring and are tolerated by the higher-level societal or corporate system in much the same way as malfunctions are tolerated by a computer system; namely by higher-level replication or through a recovery block architecture.

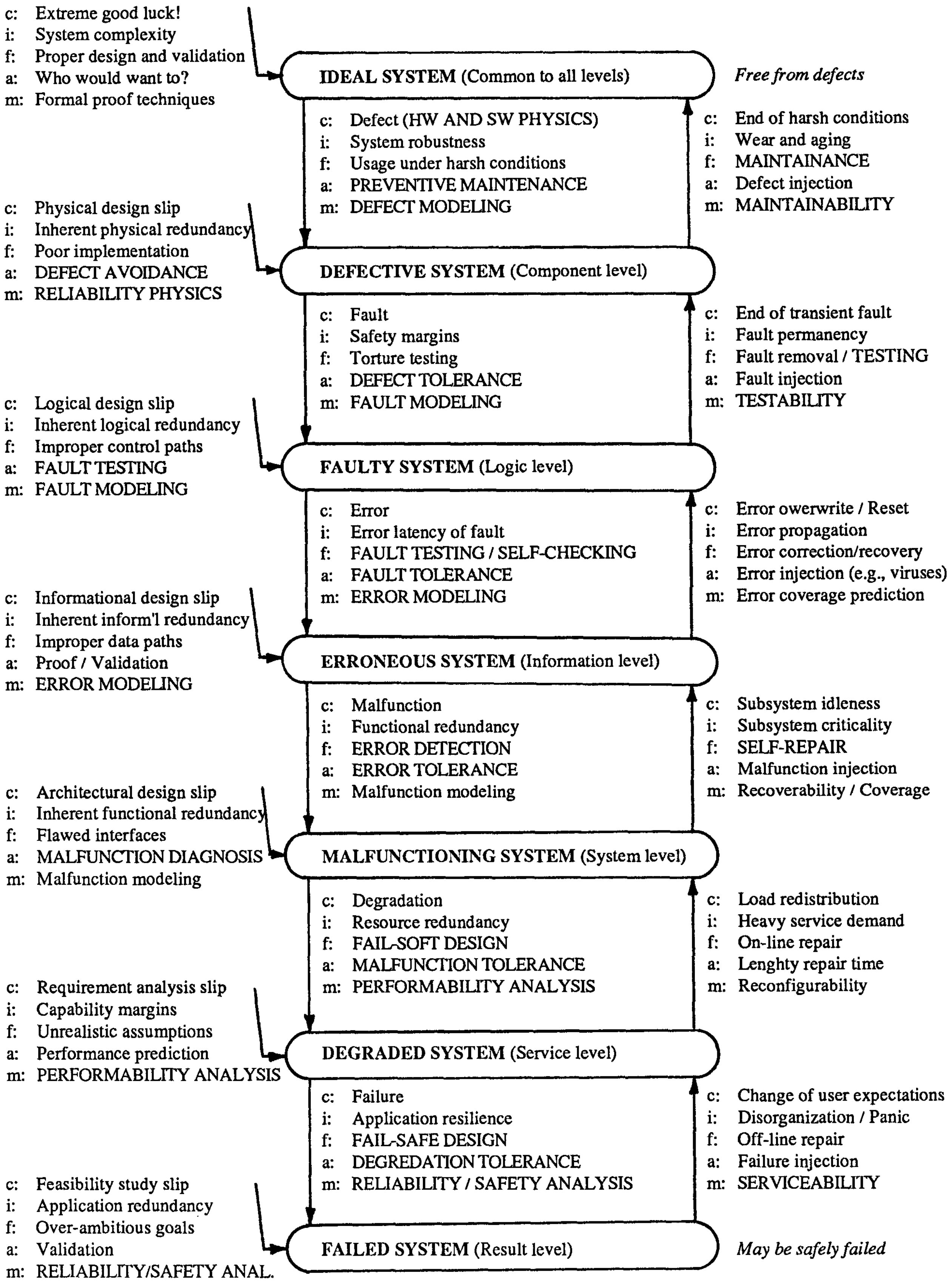

Figure 2. System States, State Transitions, and Subfields of Dependable Computing.

6. MEASURES OF DEPENDABILITY

The reliability $R(t)$, defined as the probability that a system functions properly in the time interval $[0,t]$, was the only dependability measure of interest to early designers of dependable computer systems. Such systems were typically used for spacecraft guidance and control where repairs were impossible and the system was effectively lost upon the first failure. Thus, the reliability $R(T)$ for the mission duration T accurately reflected the probability of successfully completing the mission and thus achieving acceptable reliabilities for large values of T (the so-called *long-life* systems) became the main challenge. Since reliability is a function of the mission duration T, mean time to failure (MTTF or MTFF, mean time to first failure) was often used as a single numeric indicator of system dependability. Even though it is true that a higher MTTF implies a higher reliability in the case of non-redundant systems, the use of MTTF is misleading when redundancy techniques are used.

The advent of time-sharing systems brought with it a concern for the continuity of computer service (the so-called *high-availability* systems) and thus minimizing the "down time" became a prime concern. Interval availability (or simply availability), defined as the fraction of time that the system is operational, was the natural dependability measure in this respect. The probability that a system is available at time t is known as its *pointwise availability* (which is the same as reliability when there is no repair). Availability is a function not only of how rarely a system fails but also of how soon it can be repaired upon failure. Thus, mean time to repair (MTTR) became important and *maintainability* was used as a qualitative descriptor for ease of repair. Clearly, availability can be computed as MTTF/(MTTF + MTTR) or equivalently MTTF/MTBF, where MTBF = MTTF + MTTR is the mean time between failures for repairable systems.

The concern with dependability soon spread from highly advanced special-purpose systems to commercial environments and terms like *serviceability, maintainability* , and several other "-ilities" became a permanent part of computer jargon. Increased system complexities and difficulties in testing for initial system verification and subsequent preventive and diagnostic maintenance led to concern for *testability.* Then came the widespread use of multiprocessors and gracefully degrading systems that did not obey the all-or-none mode of operation implicit in conventional reliability evaluation methods. Thus, *performability* was suggested as a relevant measure. The performability of a gracefully degrading system at time t depends on the set of resources available, the computational capability provided by these resources, and the "worth" associated with each capability level [MEYE80]. Again the desirability of a single numeric measure led to the suggestion of mean computation before failure (MCBF), although the use of this measure did not become as widespread as the MTTF and MTBF of the earlier era. In connection with malfunction diagnosis, various *diagnosability* measures have been defined (see the survey by Kime [KIME86]).

All of the above measures dealt with the operation and performance of the computer system (and frequently only with the hardware) rather than with the integrity and success of the computations performed. The availability measure, for example, does not distinguish between a system that experiences 30 two-minute outages per week and one that fails once per week but takes an hour to repair. Increasing dependence on transaction processing systems and safety-critical applications of computers has led to new concerns such as data integrity and safety which require *resilient* and *robust* hardware and software for their realization. Anderson [ANDE85] defines a *resilient* computing system as one that is "capable of providing dependable service to its users over a wide range of potentially adverse circumstances" and notes that dependability and robustness are the key attributes of such a system. He adds that a *robust* computer system "retains its ability to deliver service in conditions which are beyond its normal domain of operation, whether due to harsh treatment, or unreasonable service requests, or misoperation, or the impact of faults, or lack of maintenance ..." Both robustness and resilience are qualitative system attributes, although certain aspects of each can be quantified.

Safety, on the other hand, has been traditionally quantified. Leveson [LEVE86] defines *safety* as "the probability that conditions [leading] to mishaps (*hazards*) do not occur, whether or not the inteneded function is performed." Central to the quantification of safety is the notion of *risk.* The expected loss or risk associated with a particular failure is a function of both its severity and its probability. More precisely:

risk [*consequence / unit time*] = frequency [*events / unit time*] × magnitude [*consequence / event*]

For example, the approximate individual risk (early fatality probability per year) associated with motor vehicle accidents is 3×10^{-4} which is about 10 times the risk of drowning, 100 times the risk of railway accidents, and 1000 times the risk of being killed by a hurricane ([HENL81], p. 11). Individual risks below 10^{-6} per year are generally considered acceptable. Computer scientists and engineers have so far only scratched the surface of safety evaluation techniques [GOLD87] and much more work in this area can be expected in the coming years.

7. DEPENDABLE COMPUTER SYSTEMS

The 30-year history of dependable computing can be divided into three periods or "generations" that conveniently coincide with the past three decades. The 1960s can be viewed as the initiation and experimentation period. First-generation dependable machines built by pioneers of the field in the 1960s paved the way for significant progress in the following two decades. The middle period, the 1970s, can be described as the growth and diffusion period when numerous research projects dealt with both the theoretical and practical problems in dependable computing and many resulted in working prototypes. The 1980s have seen widespread practical application of the basic theories of dependable computing. This period can also be characterized by the many refinements and extensions introduced as a result of both operational experience and the emergence of challenging new problems (e.g., massively parallel processing, distributed systems, VLSI). The development of fourth-generation dependable systems of the 1990s will no doubt benefit from these ongoing investigations.

Even though a fault-tolerant computer called SAPO was designed and built in the early 1950s in Prague, Czechoslovakia (see [AVIZ78], p. 1119), systematic application of redundancy techniques to the design of dependable computing systems did not start until it was established in the early 1960s that the dependability requirements of telephone switching systems and on-board spacecraft computers could not be met with conventional designs. Nothworthy among early spacecraft computers were the OAO (Orbiting Astronomical Observatory) design and the Apollo guidance computer [KUEH69]. NASA's STAR (Self-Testing-and-Repairing) computer [AVIZ71] was never incorporated in a spacecraft but its laboratory use until the mid 1970s provided a broad understanding of the issues and problems of dependable computing. Bell's No. 1 ESS (Electronic Switching System) had an initial requirement of 2 hours of down time in its 40-year lifespan [DOWN64], [TOYW78]. Although this requirement was not met even by redundant designs in the early stages, the experience accumulated from extensive use of a large number of systems gradually improved the availability of subsequent designs.

Two influential second-generation dependable computing systems were MIT's fault-tolerant multiprocessor (FTMP) and Stanford's software-implemented fault tolerance (SIFT). They represented two streams of thought and two major NASA-sponsored research projects. They also provided an interesting contrast because they were developed from the same specifications; namely, to achieve a failure rate of 10^{-9} per hour in a 10-hour flight, with maintenance intervals of hundreds of hours. MIT's design [HOPK78] used flexible hardware triplication and voting, with malfunctions detected and isolated by duplex "bus guardians" designed for a malfunction-safe mode of operation. Stanford's design [WENS78] relied on replicated software tasks executing on independent off-the-shelf computing elements and communicating through a voting protocol.

Third-generation commercial computer systems of the eighties are used in two major application areas; telephone switching and transaction processing, although the industrial control market has also been considered by August Systems [WENS81] which uses the SIFT approach. In the area of telephone switching, Bell's local ESS processors have gone through many design changes over the past quarter of a century (see [ATTJ85] for description of a modern ESS processor). Some of these changes were necessitated by the changing technology while others resulted from field experience with previous designs. In the area of transaction processing, the approaches taken by Tandem Computers Inc. and Stratus Computer Inc. provide interesting examples. Tandem's NonStop systems [KATZ77], [SERL84] use redundant resources and access paths for the tolerance of hardware malfunctions, with coding and monitoring as the main detection mechanisms and periodic checkpointing as the main recovery mechanism. Stratus, whose Stratus/32 system is also marketed as IBM System/88 [HARR87], follows a design technique for processor and main memory modules that has come to be known as "pair-and-spare" [SERL84]. Essentially, a pair of identical units with an output comparator are packaged in a single customer-replaceable board and provided with a concurrently operating spare that is tightly synchronized with it, thus requiring no special recovery software.

Factors that will affect the design of fourth-generation dependable computer systems in the 1990s can be classified as technological, architectural, and functional. Technological factors refer to the special problems caused and opportunities provided by new or advancing technologies. New architectures and architectural features are the second source of challenge to researchers in the field of dependable computing. For example, massively parallel [NEGR86] and distributed systems [DOLE87] provide numerous opportunities and pitfalls. Dependability issues for dataflow and other non-von Neumann architectures need to be studied in greater depth in view of the potential importance of such systems in future. Finally, from a functional point of view, the increased complexity of future computer systems will necessitate changes in the need for and our attitude towards dependability procurement techniques.

8. CONCLUSIONS

A unified framework was provided for the study of dependable computing. It was shown that dependability procurement techniques can be applied at several levels to supplement and strengthen one another and that such a multi-level view provides a better understanding of both the impairments to dependability and the consequences of undependability. Over the years, emphasis has shifted from dependability procurement techniques at the lower levels of our model (defects, faults, errors) to the higher levels (malfunctions, degradations, failures). Therefore, we can expect dependable computers of future to use avoidance and removal techniques at the lower levels plus tolerance techniques at the higher levels to achieve their goals. This, however, does not mean that researchers will become disinterested in the lower levels altogether. Much work needs to be done in the areas of defect and fault modeling and interest will probably remain high in the use of defect and fault tolerance techniques as methods for VLSI yield enhancement [MOOR86].

In spite of significant progress made in the field of dependable computing over the past three decades, the field is still full of interesting and challenging open problems and much work remains to be done. Incorporation of dependability enhancement features will become routine practice in the design of future computer systems and the question facing the designers of such systems will become how to use a multitude of seemingly unrelated techniques in an optimal and coherent fashion to achieve given dependability goals. Methodologies for guiding the designers of hardware and software systems in their search for useful tools and strategies in a vast database of dependability procurement techniques are badly needed. It is hoped that the framework proposed in this paper will contribute to clear formulation of such methodologies and their areas of applicability in future.

Advances in VLSI are likely to result in cost-effective methods for incorporating dependability features into standard building blocks (chips) that can then be used to implement highly dependable computer systems through special synthesis procedures. Similar advances can be predicted for software technology in the form of reusable software modules with special dependability "features." New types of computing systems and related technologies (e.g., optical computing, multiple-valued logic, neural nets, etc.) all have significant dependability implications that will no doubt be explored in depth over the next decade. Massively parallel systems, which on the surface appear to be supportive of low-redundnacy dependability procurement techniques due to the possibility of shared spare resources, present difficult design problems if an inflexible interconnection topology is used [NEGR86]. Distributed systems, while avoiding some of the pitfalls associated with tightly-coupled parallel systems, present difficult problems of their own in the form of sensitivity to certain malfunction patterns and the difficulty of reaching agreement in the presence of malfunctions [DOLE87]. Dependability issues for dataflow and other non-von Neumann architectures need to be studied in greater depth [GAUD85]. There is widespread belief that our ability to provide dependability guarantees diminishes rapidly with increased complexity. This is demonstrated most dramatically by the current debate over the feasibility of implementing trustworthy computerized weapons and defense systems that can never be subjected to rigorous tests under actual operating conditions. Computer scientists and engineers are learning much from such debates [BORN87], [SOFT??] whose continuation is essential.

It is the author's belief that dependable computing can no longer be treated as a separate discipline. Rather, the useful techniques of the field must be systematically incorporated into the other areas of computer science and engineering (e.g., testability into logic design [MCCL86], reconfiguration strategies into computer architecture, performability analysis into system evaluation methodologies, etc.). In other words, the field of dependable computing now seems mature enough to disappear and be replaced by dependability subfields within the various computing subjects and disciplines. Clearly, this change of view must start with educational programs. Taking an isolated course on dependable computing encourages the computing professionals of the future to think of dependability as an add-on feature. Industrial quality control suffered from a similar problem until it was realized that quality control and the rest of manufacturing processes are inseparable.

ACKNOWLEDGEMENTS

The work reported here was initiated when the author was a Visiting Professor at the Department of Computer Science, University of Waterloo, Canada, where it was supported in part by the Natural Sciences and Engineering Research Council of Canada under grants G1140 and A5515. The current draft was completed at Carleton University, Ottawa, Canada, where the author was a Visiting Professor before joining UCSB in the Fall of 1988.

REFERENCES

The references appearing below include all published books and comprehensive survey papers in the area of dependable computing known to the author. Books are marked by asterisks and survey papers by heavy dots in the left margin.

* [ANDE81] Anderson, T. and P.A. Lee, *Fault Tolerance: Principles and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[ANDE82] Anderson, T. and P.A. Lee, "Fault Tolerance Terminology Proposals," in [FTCS82], pp. 29-33. Also in [SHRI85], pp. 6-13.

* [ANDE85] Anderson, T.A., *Resilient Computing Systems*, Collins, London, 1985. Also: Wiley, New York, 1986.

* [ARAZ87] Arazi, B., *A Commonsense Approach to the Theory of Error-Correcting Codes*, The MIT Press, 1987.

* [ARSE80] Arsenault, J.E. and J.A. Roberts (Editors), *Reliability and Maintainability of Electronic Systems*, Computer Science Press, Potomac, MD, 1980.

[ATTJ85] *AT&T Technical Journal*, Special Issue on the 5ESS Switching System, Vol. 64, No. 6, Part 2, July/Aug. 1985.

[AVIZ67] Avizienis, A., "Design of Fault-Tolerant Computers," *AFIPS Conf. Proc.*, Vol. 31 (1967 Fall Joint Computer Conf.), pp. 733-743, 1967.

[AVIZ71] Avizienis, A., G.C. Gilley, F.P. Mathur, D.A. Rennels, J.A. Rohr, and D.K. Rubin, "The STAR (Self-Testing-And-Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design," in [TRAN71], pp. 1312-1321. Reprinted in [SIEW82], pp. 523-539, and in [NELS87], pp. 107-116.

• [AVIZ78] Avizienis, A., "Fault-Tolerance: The Survival Attribute of Digital Systems," in [PROC78], pp. 1109-1125.

[AVIZ82] Avizienis, A., "The Four-Universe Information System Model for the Study of Fault Tolerance," in [FTCS82], pp. 6-13. Reprinted in [TIMO84], pp. 27-33.

[AVIZ82a] Avizienis, A., "Design Diversity -- The Challenge of the Eighties," in [FTCS82], pp. 44-45.

* [AVIZ88] Avizienis, A., H. Kopetz, and J.-C. Laprie (Editors), *Dependable Computing and Fault-Tolerant Systems: Vol. 1 -- The Evolution of Fault-Tolerant Computing*, Springer-Verlag, Wien, Austria, 1988.

* [BEIZ84] Beizer, B., *Software System Testing and Quality Assurance*, Van Nostrand, New York, 1984.

* [BHAR87] Bhargava, B.K. (Editor), *Concurrency Control and Reliability in Distributed Systems*, Van Nostrand Reinhold, New York, 1987.

• [BORN87] Borning, A., "Computer System Reliability and Nuclear War," *Communications of the ACM*, Vol. 30, No. 2, pp. 112-131, Feb. 1987.

* [BREU75] Breuer, M.A. and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Woodland Hills, CA, 1975.

[CART82] Carter, W.C., "A Time for Reflection," in [FTCS82], p. 41.

[COMP??] *Computer*, Special Issues on Fault-Tolerant Computing; (1) Vol. 4, No. 1, Jan./Feb. 1971, (2) Vol. 13, No. 3, Mar. 1980, (3) Vol. 17, No. 8, Aug. 1984.

• [DOLE87] Dolev, D., L. Lamport, M. Pease, and R. Shostak, "The Byzantine Generals," in [BHAR87], pp. 348-369.

[DOWN64] Downing, R.W., J.S. Nowak, and L.S. Tuomenoska, "No. 1 ESS Maintenance Plan," *Bell System Tecnical Journal*, Vol. 43, No. 5, Part 1, pp. 1961-2019, Sep. 1964.

* [DUNN82] Dunn, R. and R. Ullman, *Quality Assurance for Computer Software*, McGraw-Hill, New York, 1982.

[EJCC53] *Proc. of the Eastern Joint Computer Conf.* (Information Processing Systems -- Reliability and Requirements), Washington, DC, Dec. 1953.

[FORS28] Forster, E.M., "The Machine Stops," in *The Eternal Moment* (Collection of Short Stories), Harcourt Brace, 1928.

[FTCS??] *Proc. of the International Symposia on Fault-Tolerant Computing*, Held annually (usually in June) since 1971, Order from IEEE Computer Society Order Dept., PO Box 80452, Worldway Postal Center, Los Angeles, CA 90080; (1) Pasadena, CA, 1-3 Mar. 1971, (2) Newton, MA, 19-21 June 1972, (3) Palo Alto, CA, 20-22 June 1973, (4) Champaign, IL, 19-21 June 1974, (5) Paris, 18-20 June 1975, (6) Pittsburgh, PA, 21-23 June 1976, (7) Los Angeles, 28-30 June 1977, (8) Toulouse, France, 21-23 June 1978, (9) Madison, WI, 20-22 June 1979, (10) Kyoto, Japan, 1-3 Oct. 1980, (11) Portland, Maine, 24-26 June 1981, (12) Santa Monica, CA, 22-24 June 1982, (13) Milano, Italy, 28-30 June 1983, (14) Kissimmee, FL, 20-22 June 1984, (15) Ann Arbor, MI, 19-21 June 1985, (16) Vienna, Austria, 1-3 July 1986, (17) Pittsburgh, PA, 6-8 July 1987, (18) Tokyo, 27-30 June 1988.

*[FUJI85] Fujiwara, H., *Logic Testing and Design for Testability*, The MIT Press, Cambridge, MA, 1985.

[GALL77] Gall, J., *Systemantics: How Systems Work and Especially How They Fail*, Quadrangle, New York, 1977.

[GAUD85] Gaudiot, J.L. and C.S. Raghavendra, "Fault Tolerance and Data-Flow Systems," *Proc. of the International Conf. on Distributed Computing Systems*, San Francisco, May 1985, pp. 16-23.

[GOLD87] Goldberg, J., "Some Principles and Techniques for Designing Safe Systems," *Software Engineering Notes*, ACM SIGSOFT Publication, Vol. 12, No. 3, pp. 17-19, July 1987.

*[HALS77] Halstead, M.H., *Elements of Software Science*, Elsevier, New York, 1977.

[HARR87] Harrison, E.S. and E.J. Schmitt, "The Structure of System/88, A Fault-Tolerant Computer," *IBM Systems Journal*, Vol. 26, No. 3, pp. 293-318, 1987.

[HENL81] Henley, E.J. and H. Kumamoto, *Reliability Engineering and Risk Assessment*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[HOPK78] Hopkins, A.L. Jr., T.B. Smith III, and J.H. Lala, "FTMP -- A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," in [PROC78], pp. 1221-1239. Reprinted in [SIEW82], pp. 585-620.

[HOSF60] Hosford, J.E., "Measures of Dependability," *Operations Research*, Vol. 8, No. 1, pp. 53-64, Jan./Feb. 1960.

[KATZ77] Katzman, J.A., "A Fault-Tolerant Computing System," Tandem Computers, Cupertino, CA, 1977. Reprinted in [SIEW82], pp. 435-452.

•[KIME86] Kime, C.R., "System Diagnosis," Chapter 8 in [PRAD86], pp. 577-632.

[KOPE82] Kopetz, H., "The Failure Fault (FF) Model," in [FTCS82], pp. 14-17.

*[KRAF81] Kraft, G.D. and W.N. Toy, *Microprogrammed Control and Reliable Design of Small Computers*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

•[KUEH69] Kuehn, R.E., "Computer Redundancy: Design, Performance, and Future," *IEEE Transactions on Reliability*, Vol. R-18, No. 1, pp. 3-11, Feb. 1969.

•[KUHL86] Kuhl, J.G. and S.M. Reddy, "Fault-Tolerance Considerations in Large, Multiple Processor Systems," *Computer*, Vol. 19, No. 3, pp. 56-67, Mar. 1986.

*[LALA85] Lala, P.K., *Fault Tolerant and Fault Testable Hardware Design*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[LAPR82] Laprie, J.-C., "Dependability: A Unifying Concept for Reliable Computing," in [FTCS82], pp. 18-21.

[LAPR85] Laprie, J.-C., "Dependable Computing and Fault Tolerance: Concepts and Terminology," in [FTCS85], pp. 2-11.

•[LEVE86] Leveson, N.G., "Software Safety: Why, What, and How?" *Computing Surveys*, Vol. 18, No. 2, pp. 125-163, June 1986.

[LEWI87] Lewis, E.E., *Introduction to Reliability Engineering*, Wiley, New York, 1987.

*[MCCL86] McCluskey, E.J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, 1986.

*[MEYE76] Meyers, G.J., *Software Reliability: Principles and Practices*, Wiley, New York, 1976.

[MEYE80] Meyer, J.F., "On Evaluating the Performability of Degradable Computing Systems," *IEEE Transactions on Computers*, Vol. C-29, No. 8, pp. 720-731, Aug. 1980.

[MOOR56] Moore, E.F. and C.E. Shannon, "Reliable Circuits Using Less Reliable Relays," *Journal of the Franklin Institute*, Vol. 262, Part I, No. 3, pp. 191-208, Sep. 1956, and Part II, No. 4, pp. 281-297, Oct. 1956.

•[MOOR86] Moore, W.R., "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield," in [PROC86], pp. 684-698.

*[MOSS85] Moss, M.A., *Designing for Minimal Maintenance Expense*, Marcel Dekker, New York, 1985.

•[NEGR86] Negrini, R., M. Sami, and R. Stefanelli, "Fault Tolerance Techniques for Array Structures Used in Supercomputing," *Computer*, Vol. 19, No. 2, pp. 78-87, Feb. 1986. Reprinted in [NELS87], pp. 403-412.

*[NELS87] Nelson, V.P. and B.D. Carroll (Editors), *Tutorial: Fault-Tolerant Computing*, IEEE Computer Society Press, Washington, DC, 1987.

•[PARH78] Parhami, B., "Errors in Digital Computers: Causes and Cures," *Australian Computer Bulletin*, Vol. 2, No. 2, pp. 7-12, Mar. 1978.

•[PARH78a] Parhami, B., "Fault-Tolerant Digital System Hardware: Introduction and Overview," *Proc. of the International Conf. on Measurement and Control*, Athens, pp. 883-889, June 1978.

•[PARH88] Parhami, B., "A Multi-Level View of Dependable Computing," Submitted for publication.

*[PETE72] Peterson, W.W. and E.J. Weldon Jr., *Error-Correcting Codes*, MIT Press, Cambridge, MA, 2nd Edition, 1972.

*[PIER65] Pierce, W.H., *Failure-Tolerant Computer Design*, Academic Press, New York, 1965.

* [PRAD86] Pradhan, D.K. (Editor), *Fault-Tolerant Computing: Theory and Techniques*, 2 Vols., Prentice-Hall, 1988.

[PROC78] *Proceedings of the IEEE*, Special Issue on Fault-Tolerant Computing, Vol. 66, No. 10, pp. 1107-1273, Oct. 1978.

[PROC86] *Proceedings of the IEEE*, Special Issue on Fault Tolerance in VLSI, Vol. 74, No. 5, May 1986.

• [RAND78] Randell, B., P.A. Lee, and P.C. Treleaven, "Reliability Issues in Computing System Design," *Computing Surveys*, Vol. 10, No. 2, pp. 123-165, June 1978.

* [RAOT74] Rao, T.R.N., *Error Codes for Arithmetic Processors*, Academic Press, New York, 1974.

• [RENN84] Rennels, D.A., "Fault-Tolerant Computing -- Concepts and Examples," *IEEE Transactions on Computers*, Vol. C-33, No. 12, pp. 1116-1129, Dec. 1984.

• [SERL84] Serlin, O., "Fault-Tolerant Systems in Commercial Applications," in [COMP84], pp. 19-30. Reprinted in [NELS87], pp. 19-30.

• [SHOR68] Short, R.A., "The Attainment of Reliable Digital Systems Through the Use of Redundancy -- A Survey," *IEEE Computer Group News* (now *Computer*), Vol. 2, No. 2, pp. 2-17, Mar. 1968.

* [SHRI85] Shrivastava, S.K. (Editor), *Reliable Computer Systems: Collected Papers of the Newcastle Reliability Project*, Springer-Verlag, Berlin, 1985.

* [SIEW82] Siewiorek, D.P. and R.S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, MA, 1982.

[SOFT??] *Software Engineering Notes*, ACM SIGSOFT Publication, contains regular feature entitled "Forum on Risks to the Public in Computer Systems."

* [TIMO84] Timoc, C.C. (Editor), *Selected Reprints on Logic Design for Testability*, IEEE Computer Society Press, Silver Spring, MD, 1984.

[TOYW78] Toy, W.N., "Fault-Tolerant Design of Local ESS Processors," in [PROC78], pp. 1126-1145. Reprinted in [SIEW82], pp. 461-496.

[TRAN??] *IEEE Transactions on Computers*, Special Issues or Sections on fault-Tolerant Computing; (1) Vol. C-20, No. 11, Nov. 1971, (2) Vol. C-22, No. 3, Mar. 1973, (3) Vol. C-23, No. 7, July 1974, (4) Vol. C-24, No. 5, May 1975, (5) Vol. C-25, No. 6, June 1976, (6) Vol. C-27, No. 6, June 1978, (7) Vol. C-29, No. 6, June 1980, (8) Vol. C-31, No. 7, July 1982, (9) Vol. C-33, No. 6, June 1984, (10) Vol. C-35, No. 4, Apr. 1986, (11) Vol. 37, No. 4, Apr. 1988.

* [VOGE88] Voges, U. (Editor), *Dependable Computing and Fault-Tolerant Systems: Vol. 2 -- Software Diversity in Computerized Control Systems*, Springer-Verlag, Wien, Austria, 1988.

[VONN56] von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," in *Automata Studies* (Annals of Mathematics Studies, No. 34), Ed. by C.E. Shannon and J. McCarthy, Princeton Univ. Press, pp. 43-98, 1956. Reprinted in *John von Neumann: Collected Works*, Ed. by A.H. Taub, Pergamon Press, New York, Vol. V, pp. 329-378, 1963.

* [WAKE78] Wakerly, J.F., *Error Detecting Codes, Self-Checking Circuits and Applications*, North-Holland, New York, 1978.

[WENS78] Wensley, J.H., L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shostak, and C.B. Weinstock, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," in [PROC78], pp. 1240-1255. Reprinted in [SIEW82], pp. 559-584, and in [NELS87], pp. 145-160.

[WENS81] Wensley, J.H., "Fault-Tolerant Computers Ensure Reliable Industrial Controls," *Electronic Design*, Vol. 29, No. 13, 25 June 1981.

* [WILC62] Wilcox, R.H. and W.C. Mann (Editors), *Redundancy Techniques for Computing Systems*, Spartan, Washington, DC, 1962.

* [WILL86] Williams, T.W. (Editor), *VLSI Testing*, North-Holland, Amsterdam, 1986.

• [WILL86a] Williams, T.W., "Design for Testability," Chapter 4 in [WILL86], pp. 95-160.

* [WINO63] Winograd, S. and J.D. Cowan, *Reliable Computation in the Presence of Noise*, MIT Press, Cambridge, MA, 1963.

• [ZORP85] Zorpette, G., "Computers that Are 'Never' Down," *IEEE Spectrum*, Vol. 22, No. 4, pp. 46-54, Apr. 1985.