A FRAMEWORK FOR THE STUDY OF COMPUTER SYSTEM DEPENDABILITY

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 system abstraction levels corresponding to defects, faults, errors, malfunctions, degradations, and failures. It is argued that all of these levels are useful, in the sense that proven dependability assurance 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 each level.

1. BACKGROUND

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 concept reflected for example in engineers' "safety factor" and accountants' replicated manual or mechanical calculations. 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 computation results 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 highly complex, intelligent computers for the future, computer system dependability should become an integral part of the design process for future generation systems. A general framework within which research studies in the area of computer system dependability can be conducted, presented, discussed, and classified will not only facilitate this integration but also lead to a better understanding of the concepts and techniques of the field.

2. AN EVOLVING DEFINITION

A prerequisite for progress and effective communication in any area of study is a set of precise, widely accepted definitions for the field and its fundamental concepts. Many attempts have been made at defining (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 incapacitated. The problem lies in the phrase "be able to operate." What we are actually interested in is "task accomplishment" rather than "system operation." The following definition, which is due to Laprie [LAPR82], is more suitable in this respect:

"... 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 (which, by the way, is common practice!). Carter provides a much 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 [it]."

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

23ACSSC-12/89/1017 $1.00 © 1989 MAPLE PRESS

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/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. Thus we are led to:

Proposal: *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.*

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 5 of this paper.

3. A MULTI-LEVEL VIEW

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 viewing and describing such impairments.

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). This recursion stops when we arrive at atomic systems whose internal structures are of little or no interest at the 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."*

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	Undesired Event
Physical	Failure
⇓	⇓
Logical	Fault
⇓	⇓
Informational	Error
⇓	⇓
External	Crash

In what follows, we extend and refine this 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 we expect from 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 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 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	Impairment
Component	Defect
⇓	⇓
Logic	Fault
⇓	⇓
Information	Error
⇓	⇓
System	Malfunction
⇓	⇓
Service	Degradation
⇓	⇓
Result	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 thus be consistent by always associating the term "failure" with the highest and the term "defect" with the lowest level of abstraction.

Figure 1 provides an analogy for accentuating the chain of cause-effect relationships in our multi-level model. 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 is analogous to a computer failure affecting the larger corporate or societal system.

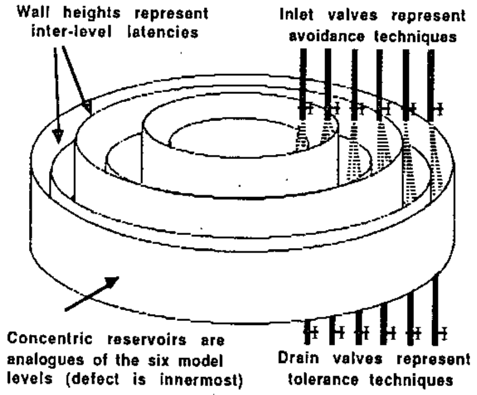
Wall heights represent inter-level latencies
Inlet valves represent avoidance techniques
Concentric reservoirs are analogues of the six model levels (defect is innermost)
Drain valves represent tolerance techniques

Figure 1. A simple analogy for our multi-level model of dependable computing. Flow of water represents defects, faults, errors, malfunctions, degradations, and failures.

4. USE OF THE MODEL

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 has 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 in Figure 2 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 in Figure 2 are five properties that can be specified as transition labels or tags:

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 some of the tags can be the basis of new studies and subfields. In addition such transition tags can be used for classifying or indexing of techniques and research studies in the field of dependable computing. For example, a research project or proposed redundancy technique 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 [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.

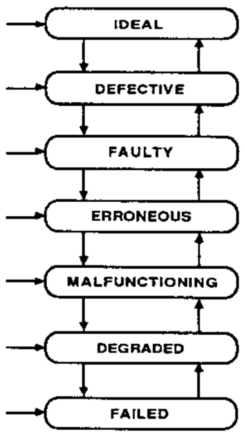

Figure 2. System states and state transitions in the multi-level model of dependable computing.

The unifying influence of this model is that it allows us to think of all known dependability procurement techniques as being related to initially *avoiding* a particular undesired state, forcing a transition to a higher state by *removing* the underlying cause, and *tolerating* particular undesired events by structuring the system so that transition to a lower state is severly impeded. Thus, the three strategies of *avoidance*, *removal*, and *tolerance* for the design of dependable systems actually correspond to starting the system at the highest state that is practically feasible, facilitating upward transitions corresponding to remedies, and impeding downward transitions resulting from deviations.

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 [ANDE81], [AVIZ78], [COMPyr], [FTCSyr], [JOHN89], [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 replication-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.

Modeling studies and tools can also be related to the multi-level model of Figure 2 in a natural way. *Defect modeling* deals with the transition from ideal to defective state, while *fault modeling* relates physical defects to logic faults. The transition from malfunctioning to degraded state is the subject of *performability analysis* methods. Finally, *Safety analysis* deals with the terminal transition into the failed state and its consequences.

5. DISCUSSION

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 [AVIZ88]. 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).

One may observe that 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 future dependable computers 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 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 unifying framework presented in this paper along with a flexible strategy for dependability assurance suggested elsewhere [PARH89], [PARH89a] will contribute to clear formulation of such methodologies and their areas of applicability in future.

REFERENCES

[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.

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

[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.

[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, 1988.

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

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

[COMPyr] *Computer*, Special Issues on Fault-Tolerant Computing.
(71) Vol. 4, No. 1, Jan./Feb. (80) Vol. 13, No. 3, Mar.
(84) Vol. 17, No. 8, Aug. (90) Vol. 23, No. 7, July (?)

[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.

[FTCSyr] *Proc. of the Int'l Symp. on Fault-Tolerant Computing*, Held annually (usually in June) since 1971, Order from IEEE Computer Society. Year of the symposium is given in parentheses.
(71) Pasadena, CA, Mar. 1-3 (72) Newton, MA, June 19-21
(73) Palo Alto, CA, June 20-22 (74) Champaign, IL, June 19-21
(75) Paris, June 18-20 (76) Pittsburgh, PA, June 21-23
(77) Los Angeles, June 28-30 (78) Toulouse, France, June 21-23
(79) Madison, WI, June 20-22 (80) Kyoto, Japan, Oct. 1-3
(81) Portland, Maine, June 24-26 (82) Santa Monica, CA, June 22-24
(83) Milano, Italy, June 28-30 (84) Kissimmee, FL, June 20-22
(85) Ann Arbor, MI, June 19-21 (86) Vienna, Austria, July 1-3
(87) Pittsburgh, PA, July 6-8 (88) Tokyo, June 27-30
(89) Chicago, June 21-23 (90) Newcastle/Tyne, UK,June 26-28

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

[JOHN89] Johnson, B.W., *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, 1989.

[KATZ77] Katzman, J.A., "A Fault-Tolerant Computing System," Tandem Computers, Cupertino, CA, 1977.

[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.

[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., 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.

[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.

[PARH88] Parhami, B., "A Multi-Level View of Dependable Computing," Submitted to *IEEE Transactions on Computers*.

[PARH89] Parhami, B., "A New Paradigm for the Design of Dependable Systems," *Proc. of the Int'l Symp. on Circuits and Systems*, Portland, OR, May 1989, pp. 561-564.

[PARH89a] Parhami, B., "A Data-Driven Dependability Assurance Scheme with Applications to Data and Design Diversity," *Proc. of the IFIP Int'l Working Conf. on Dependable Computing for Critical Applications*, Santa Barbara, CA, Aug. 1989, pp. 105-112.

[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.

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

[TRANyr] *IEEE Transactions on Computers*, Special Issues or Sections on Fault-Tolerant Computing;
(71) Vol. C-20, No. 11, Nov. (73) Vol. C-22, No. 3, Mar.
(74) Vol. C-23, No. 7, July (75) Vol. C-24, No. 5, May
(76) Vol. C-25, No. 6, June (78) Vol. C-27, No. 6, June
(80) Vol. C-29, No. 6, June (82) Vol. C-31, No. 7, July
(84) Vol. C-33, No. 6, June (86) Vol. C-35, No. 4, Apr.
(88) Vol. 37, No. 4, Apr. (90) Vol. 39, No. 4, Apr. (?)

[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.

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