Fundamental Concepts of Dependability

Algirdas Avižienis
UCLA, Los Angeles, CA, USA
and Vytautas Magnus U.
Kaunas, Lithuania
aviz@cs.ucla.edu

Jean-Claude Laprie
LAAS-CNRS
Toulouse
France
laprie@laas.fr

Brian Randell
Dept. of Computing Science
U. of Newcastle upon Tyne
U.K.
Brian.Randell@newcastle.ac.uk

1. Origins and Integration of the Concepts

The concept of dependable computing first appears in the 1830's in the context of Babbage's Calculating Engine [1,2]. The first generation of electronic computers (late 1940's to mid-50's) used rather unreliable components, therefore practical techniques were employed to improve their reliability, such as error control codes, duplexing with comparison, triplication with voting, diagnostics to locate failed components, etc. [3-5].

At the same time J. von Neumann [6], E. F. Moore and C. E. Shannon [7] and their successors developed theories of using redundancy to build reliable logic structures from less reliable components, whose faults were masked by the presence of multiple redundant components. The theories of masking redundancy were unified by W. H. Pierce as the concept of failure tolerance in 1965 [8]. In 1967, A. Avizienis integrated masking with the practical techniques of error detection, fault diagnosis, and recovery into the concept of fault-tolerant systems [9]. In the reliability modeling field, the major event was the introduction of the coverage concept by Bouricius, Carter and Schneider [10]. Seminal work on software fault tolerance was initiated by B. Randell [11,12], later it was complemented by N-version programming [13].

The formation of the IEEE-CS TC on Fault-Tolerant Computing in 1970 and of IFIP WG 10.4 *Dependable Computing and Fault Tolerance* in 1980 accelerated the emergence of a consistent set of concepts and terminology. Seven position papers were presented in 1982 at FTCS-12 [14], and J.-C. Laprie formulated a synthesis in 1985 [15]. Further work by members of IFIP WG 10.4, led by J.-C. Laprie, resulted in the 1992 book *Dependability: Basic Concepts and Terminology* [16], in which the English text was also translated into French, German, Italian, and Japanese.

In [16], intentional faults (malicious logic, intrusions) were listed along with accidental faults (physical, design, or interaction faults). Exploratory research on the integration of fault tolerance and the defenses against the intentional faults, i.e., security threats, was started at the RAND Corporation [17], University of Newcastle [18], LAAS [19], and UCLA [20,21] in the mid-80's. The 1st IFIP Working Conference on Dependable Computing for Critical Applications was held in 1989. This and the six working conferences that followed fostered the interaction of the dependability and security communities, and advanced the integration of security (confidentiality, integrity and availability) into the framework of dependable computing [22]. A summary of [22] is presented next.

The alphabetic ordering of authors' names does not imply any seniority of authorship

2. The Principal Concepts: a Summary

A systematic exposition of the concepts of dependability consists of three parts: the **threats** to, the **attributes** of, and the **means** by which dependability is attained, as shown in Figure 1.

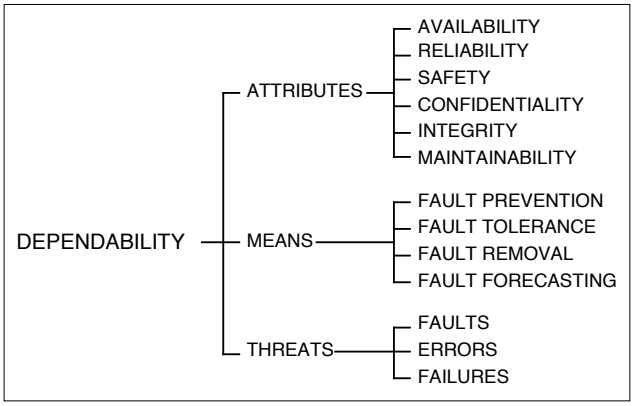

Figure 1 - The dependability tree

Computing systems are characterized by four fundamental properties: functionality, performance, cost, and dependability. **Dependability** of a computing system is the ability to deliver service that can justifiably be trusted. The **service** delivered by a system is its behavior as it is perceived by its user(s); a **user** is another system (physical, human) that interacts with the former at the **service interface**. The **function** of a system is what the system is intended for, and is described by the system specification.

2.1. The Threats: Faults, Errors, and Failures

Correct service is delivered when the service implements the system function. A system **failure** is an event that occurs when the delivered service deviates from correct service. A system may fail either because it does not comply with the specification, or because the specification did not adequately describe its function. A failure is a transition from correct service to **incorrect**

service, i.e., to not implementing the system function. A transition from incorrect service to correct service is **service restoration**. The time interval during which incorrect service is delivered is a service **outage**. An **error** is that part of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service. A **fault** is the adjudged or hypothesized cause of an error. A fault is **active** when it produces an error, otherwise it is **dormant**.

A system does not always fail in the same way. The ways a system can fail are its **failure modes**. As shown in Figure 2, the modes characterize incorrect service according to three viewpoints: a) the failure domain, b) the perception of a failure by system users, and c) the consequences of failures on the environment.

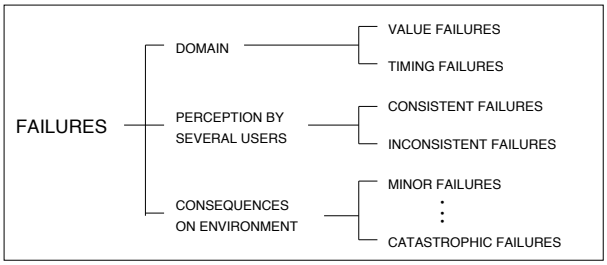

Figure 2 - The failure modes

A system consists of a set of interacting components, therefore the system state is the set of its component states. A fault originally causes an error within the state of one (or more) components, but system failure will not occur as long as the error does not reach the service interface of the system. A convenient classification of errors is to describe them in terms of the component failures that they cause, using the terminology of Figure 2: value vs. timing errors; consistent vs. inconsistent ('Byzantine') errors when the output goes to two or more components; errors of different severities: minor vs. ordinary vs. catastrophic errors. An error is **detected** if its presence in the system is indicated by an **error message** or **error signal** that originates within the system. Errors that are present but not detected are **latent** errors.

Faults and their sources are very diverse. Their classification according to six major criteria is presented in Figure 3. Of special interest for this Workshop are faults that result from attacks on a system. They are classified as deliberately malicious (d.m.) faults and include malicious logic [23] (Trojan horses, logic bombs, trapdoors are d.m. design faults, while viruses and worms are d.m. operational faults), intrusions (d.m. external operational faults) and physical attacks on a system (d.m. physical operational faults). Figure 4 shows the combined fault classes for which defenses need to be devised.

Fault pathology, i.e., the relationship between faults, errors, and failures is summarized by Figure 5, which gives the fundamental chain of threats to dependability. The arrows in this chain express a causality relationship between faults, errors and failures. They should be interpreted generically: by propagation, several errors can be generated before a failure occurs.

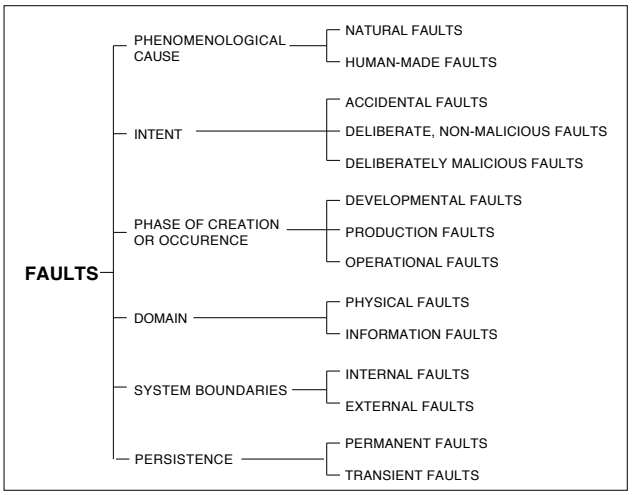

Figure 3 - Elementary fault classes

2.2. The Attributes of Dependability

Dependability is an integrative concept that encompasses the following attributes: **availability**: readiness for correct service; **reliability**: continuity of correct service; **safety**: absence of catastrophic consequences on the user(s) and the environment; **confidentiality**: absence of unauthorized disclosure of information; **integrity**: absence of improper system state alterations; **maintainability**; ability to undergo repairs and modifications.

Security is the concurrent existence of a) availability for authorized users only, b) confidentiality, and c) integrity with 'improper' meaning 'unauthorized'.

The above attributes may be emphasized to a greater or lesser extent depending on the application: availability is always required, although to a varying degree, whereas reliability, safety, confidentiality may or may not be required. The extent to which a system possesses the attributes of dependability should be interpreted in a relative, probabilistic, sense, and not in an absolute, deterministic sense: due to the unavoidable presence or occurrence of faults, systems are never totally available, reliable, safe, or secure.

Integrity is a prerequisite for availability, reliability and safety, but may not be so for confidentiality (for instance attacks via covert channels or passive listening can lead to a loss of confidentiality, without impairing integrity). The definition given for integrity – absence of improper system state alterations – extends the usual definition as follows: (a) when a system implements an authorization policy, 'improper' encompasses 'unauthorized'; (b) 'improper alterations' encompass actions resulting in preventing (correct) upgrades of information; (c) 'system state' encompasses hardware modifications or damages. The definition given for maintainability goes beyond corrective and preventive maintenance, and encompasses two other forms of maintenance: adaptive and perfective maintenance.

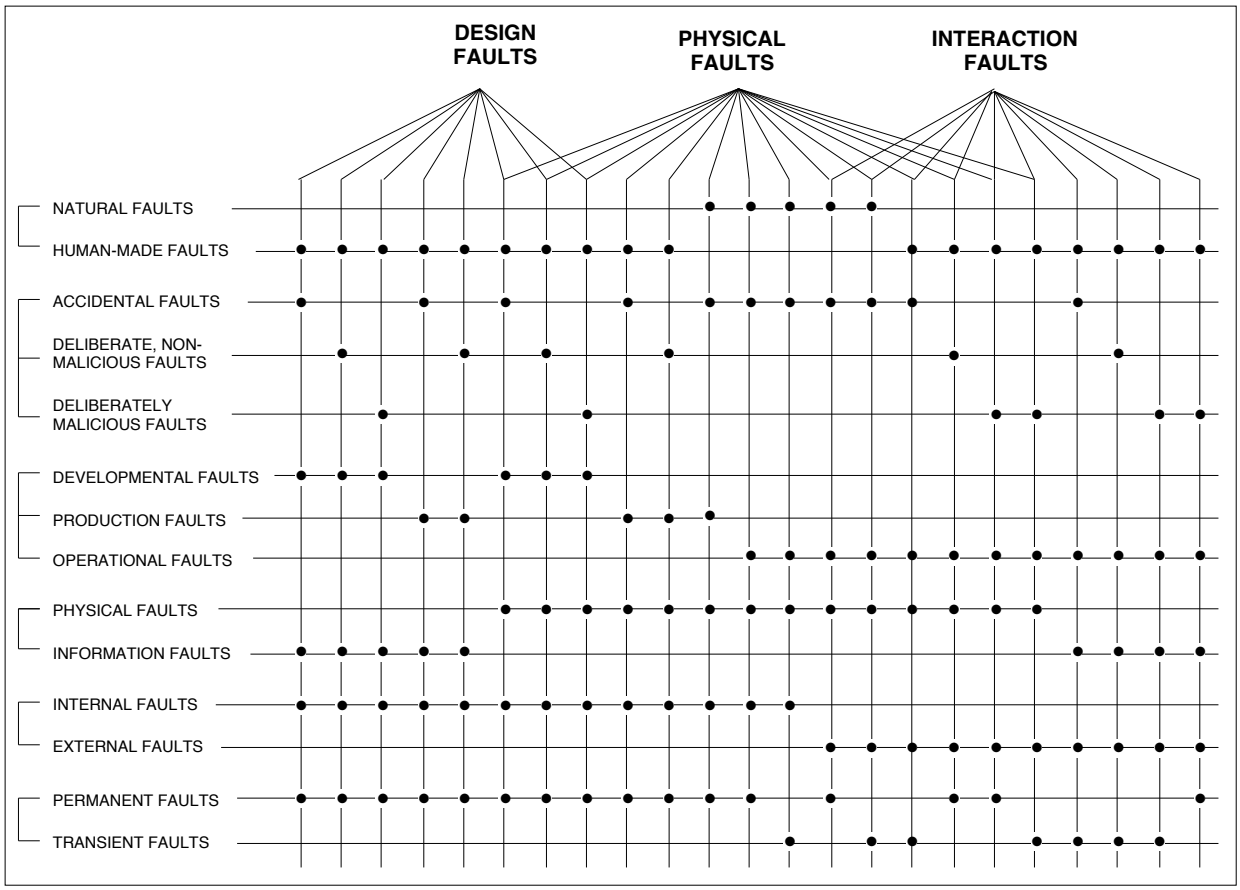

Figure 4 - Combined fault classes

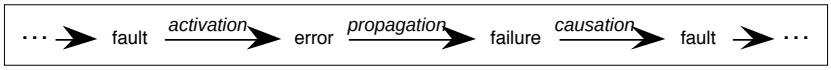

Figure 5 - The fundamental chain of threats to dependability

Security has not been introduced as a single attribute of dependability. This is in agreement with the usual definitions of security, which view it as a composite notion, namely [24] "the combination of (1) confidentiality (the prevention of the unauthorized disclosure of information), (2) integrity (the prevention of the unauthorized amendment or deletion of information), and (3) availability (the prevention of the unauthorized withholding of information)". A single definition for **security** could be: the absence of unauthorized access to, or handling of, system state.

In their definitions, availability and reliability emphasize the avoidance of failures, while safety and security emphasize the avoidance of a specific class of failures (catastrophic failures, unauthorized access or handling of information, respectively). Reliability and availability are thus closer to each other than they are to safety on one hand, and to security on the other; reliability and availability can be grouped together, and collectively defined as the avoidance or minimization of service outages.

The variations in the emphasis on the different attributes of dependability directly affect the appropriate balance of the techniques (fault prevention, tolerance, removal and forecasting) to be employed in order to make the resulting system dependable. This problem is all the more difficult as some of the attributes conflict (e.g. availability and safety, availability and security), necessitating design trade-offs.

2.3. The Means to Attain Dependability

The development of a dependable computing system calls for the combined utilization of a set of four techniques: **fault prevention**: how to prevent the occurrence or introduction of faults; **fault tolerance**: how to deliver correct service in the presence of faults; **fault removal**: how to reduce the number or severity of faults; **fault forecasting**: how to estimate the present number, the future incidence, and the likely consequences of faults.

2.3.1. Fault Prevention

Fault prevention is attained by quality control techniques employed during the design and manufacturing of hardware and software. They include structured programming, information hiding, modularization, etc.,

for software, and rigorous design rules for hardware. Operational physical faults are prevented by shielding, radiation hardening, etc., while interaction faults are prevented by training, rigorous procedures for maintenance, "foolproof" packages. Malicious faults are prevented by firewalls and similar defenses.

2.3.2. Fault Tolerance

Fault tolerance is intended to preserve the delivery of correct service in the presence of active faults. It is generally implemented by error detection and subsequent system recovery.

Error detection originates an error signal or message within the system. An error that is present but not detected is a **latent** error. There exist two classes of error detection techniques: (a) **concurrent error detection**, which takes place during service delivery; and (b) **preemptive error detection**, which takes place while service delivery is suspended; it checks the system for latent errors and dormant faults.

Recovery transforms a system state that contains one or more errors and (possibly) faults into a state without detected errors and faults that can be activated again. Recovery consists of error handling and fault handling. **Error handling** eliminates errors from the system state. It may take two forms: (a) **rollback**, where the state transformation consists of returning the system back to a saved state that existed prior to error detection; that saved state is a **checkpoint**, (b) **rollforward**, where the state without detected errors is a new state.

Fault handling prevents located faults from being activated again. Fault handling involves four steps: (a) **fault diagnosis** that identifies and records the cause(s) of error(s), in terms of both location and type, (b) **fault isolation** that performs physical or logical exclusion of the faulty components from further participation in service delivery, i.e., it makes the fault dormant, (c) **system reconfiguration** that either switches in spare components or reassigns tasks among non-failed components, (d) **system reinitialization** that checks, updates and records the new configuration and updates system tables and records. Usually, fault handling is followed by corrective maintenance that removes faults isolated by fault handling. The factor that distinguishes fault tolerance from maintenance is that maintenance requires the participation of an external agent.

The use of sufficient redundancy may allow recovery without explicit error detection. This form of recovery is called **fault masking**.

Preemptive error detection and handling (often called BIST: built-in self-test), possibly followed by fault handling is performed at system power up. It also comes into play during operation, under various forms such as spare checking, memory scrubbing, audit programs, or the so-called software rejuvenation, aimed at removing the effects of software aging before they lead to failure.

The choice of error detection, error handling and fault handling techniques, and of their implementation, is directly related to the underlying fault assumption. The classes of faults that can actually be tolerated depend on the fault assumption in the design process. A widely-used method of fault tolerance is to perform multiple computations in multiple channels, either sequentially or concurrently. When tolerance of operational physical faults is required, the channels may be of identical design, based on the assumption that hardware components fail independently. Such an approach has proven to be adequate for elusive design faults, via rollback, however it is not suitable for the tolerance of solid design faults, which necessitates that the channels implement the same function via separate designs and implementations, i.e., through **design diversity**.

Fault tolerance is a recursive concept: it is essential that the mechanisms that implement fault tolerance should be protected against the faults that might affect them. Examples are voter replication, self-checking checkers, 'stable' memory for recovery programs and data, etc. Systematic introduction of fault tolerance is facilitated by the addition of support systems specialized for fault tolerance such as software monitors, service processors, dedicated communication links.

Fault tolerance is not restricted to accidental faults. Some mechanisms of error detection are directed towards both malicious and accidental faults (e.g. memory access protection techniques) and schemes have been proposed for the tolerance of both intrusions and physical faults, via information fragmentation and dispersal, as well as for tolerance of malicious logic, and more specifically of viruses, either via control flow checking, or via design diversity.

2.3.3. Fault Removal

Fault removal is performed both during the development phase, and during the operational life of a system. Fault removal during the development phase of a system life-cycle consists of three steps: verification, diagnosis, correction. **Verification** is the process of checking whether the system adheres to given properties, termed the verification conditions. If it does not, the other two steps follow: diagnosing the fault(s) that prevented the verification conditions from being fulfilled, and then performing the necessary corrections.

Checking the specification is usually referred to as **validation**. Uncovered specification faults can happen at any stage of the development, either during the specification phase itself, or during subsequent phases when evidence is found that the system will not implement its function, or that the implementation cannot be achieved in a cost effective way.

Verification techniques can be classified according to whether or not they involve exercising the system. Verifying a system without actual execution is **static verification**. Verification a system through exercising it constitutes **dynamic verification**; the inputs supplied to the system can be either symbolic in the case of **symbolic execution**, or actual in the case of verification testing, usually simply termed **testing**. An important aspect is the verification of fault tolerance mechanisms, especially a) formal static verification, and b) testing that necessitates

faults or errors to be part of the test patterns, that is usually referred to as **fault injection**. Verifying that the system cannot do more than what is specified is especially important with respect to what the system should not do, thus with respect to safety and security. Designing a system in order to facilitate its verification is termed **design for verifiability**. This approach is well-developed for hardware with respect to physical faults, where the corresponding techniques are termed design for testability.

Fault removal during the operational life of a system is corrective or preventive maintenance. **Corrective maintenance** is aimed to remove faults that have produced one or more errors and have been reported, while **preventive maintenance** is aimed to uncover and remove faults before they might cause errors during normal operation. The latter faults include a) physical faults that have occurred since the last preventive maintenance actions, and b) design faults that have led to errors in other similar systems. Corrective maintenance for design faults is usually performed in stages: the fault may be first isolated (e.g., by a workaround or a patch) before the actual removal is completed. These forms of maintenance apply to non-fault-tolerant systems as well as fault-tolerant systems, that can be maintainable on-line (without interrupting service delivery) or off-line (during service outage).

2.3.4. Fault Forecasting

Fault forecasting is conducted by performing an evaluation of the system behavior with respect to fault occurrence or activation. Evaluation has two aspects: (1) **qualitative**, or **ordinal, evaluation**, that aims to identify, classify, rank the failure modes, or the event combinations (component failures or environmental conditions) that would lead to system failures; (2) **quantitative**, or **probabilistic, evaluation**, that aims to evaluate in terms of probabilities the extent to which some of the attributes of dependability are satisfied; those attributes are then viewed as measures of dependability. The methods for qualitative and quantitative evaluation are either specific (e.g., failure mode and effect analysis for qualitative evaluation, or Markov chains and stochastic Petri nets for quantitative evaluation), or they can be used to perform both forms of evaluation (e.g., reliability block diagrams, fault-trees).

The evolution of dependability over a system's life-cycle is characterized by the notions of stability, growth, decrease that can be stated for the various attributes of dependability. These notions are illustrated by **failure intensity**, i.e., the number of failures per unit of time. It is a measure of the frequency of system failures, as noticed by its user(s). Failure intensity typically first decreases (reliability growth), then stabilizes (stable reliability) after a certain period of operation, then increases (reliability decrease), and the cycle resumes.

The alternation of correct-incorrect service delivery is quantified to define reliability, availability and maintainability as measures of dependability: (1) **reliability**: a measure of the continuous delivery of correct service — or, equivalently, of the time to failure; (2) **availability:** a measure of the delivery of correct service with respect to the alternation of correct and incorrect service; (3) **maintainability**: a measure of the time to service restoration since the last failure occurrence, or equivalently, measure of the continuous delivery of incorrect service; (4) **safety** is an extension of reliability. When the state of correct service and the states of incorrect service due to non-catastrophic failure are grouped into a safe state (in the sense of being free from catastrophic damage, not from danger), **safety** is a measure of continuous safeness, or equivalently, of the time to catastrophic failure. Safety is thus reliability with respect to catastrophic failures.

Generally, a system delivers several services, and there often are two or more modes of service quality, e.g. ranging from full capacity to emergency service. These modes distinguish less and less complete service deliveries. Performance-related measures of dependability are usually subsumed into the notion of performability.

The two main approaches to probabilistic fault-forecasting, aimed to derive probabilistic estimates of dependability measures, are modeling and (evaluation) testing. These approaches are complementary, since modeling needs data on the basic processes modeled (failure process, maintenance process, system activation process, etc.), that may be obtained either by testing, or by the processing of failure data.

When evaluating fault-tolerant systems, the coverage provided by error and fault handling mechanisms has a drastic influence on dependability measures. The evaluation of coverage can be performed either through modeling or through testing, i.e. fault injection.

3. Relating Survivability and Dependability

The dependability concepts outlined above are the results of nearly twenty years of activity. Survivability can be traced back to the late sixties - early seventies in the military standards, where it was defined as a system capacity to resist a hostile environment so that it can fulfill its mission (see, e.g., MIL-STD-721 or DOD-D-5000.3).

Dependability has evolved from reliability/availability concerns, along with the technological developments of the computing and communications field, in order to respond adequately to the challenges posed by increasingly networked applications, and by the increase in the necessary reliance we have to place on ubiquitous computing. Survivability, as it is understood in the workshop (according to the call for participation: “ability of a system to continue to fulfill its mission in the presence of attacks, failures, or accidents”) has evolved [25] from pure security concerns; it has gained much prominence with the increase of frequency and severity of attacks by intelligent adversaries on mission-critical networked information systems.

From the perspective of the dependability framework, survivability is dependability in the presence of active faults, having in mind all the classes of faults discussed in section 2.1. However, dependability and survivability

are actually very close to each other, especially when looking at the primitives for implementing survivability, i.e., the "3 R's": Resistance, Recognition, and Recovery [25]. Resistance, i.e., the ability to repel attacks, relates, in dependability terms, to fault prevention. Recognition, i.e., the ability to recognize attacks and the extent of damage, together with Recovery, i.e., the ability to restore essential services during attack, and to recover full service after attack, have much in common with fault tolerance. Clearly dependability and survivability both go beyond the traditional approaches, based on fault avoidance, and have recognized the necessity of fault tolerance. Dependability and survivability, via independent evolutions, have actually converged and are much in agreement.

It is evident that the coordinated best efforts of dependability and survivability researchers and practitioners will be needed to devise defenses against the many threats to mission-critical systems.

Acknowledgment

We thank all colleagues who contributed to the 1992 book [16] and added their advice on desirable improvements and extensions during the writing of the current document [22]. Many thanks to Dr. Yutao He, to Jackie Furgal and Christine Fourcade, who contributed to the preparation of this text while the authors were scattered around the globe.

References

[1] D. Lardner, Babbage's calculating engine. Edinburgh Review, July 1834. Reprinted in P. Morrison and E. Morrison, editors, *Charles Babbage and His Calculating Engines*. Dover, 1961.

[2] C. Babbage. On the mathematical powers of the calculating engine (December 1837). Unpublished Manuscript. Buxton MS7, Museum of the History of Science. In B. Randell, editor, *The Origins of Digital Computers: Selected papers,* pages 17-52. Springer, 1974.

[3] *Proceedings of the Joint AIEE-IRE Computer Conference,* December 1951.

[4] Information processing systems - reliability and requirements. In *Proccedings of Eastern Joint Computer Conference,* December 1953.

[5] Session 14: Symposium: Diagnostic programs and marginal checking for large scale digital computers. In *Convention Record of the IRE 1953 National Convention, part 7*, pages 48-71, March 1953.

[6] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Annals of Math Studies*, numbers 34, pages 43-98. Princeton Univ. Press, 1956.

[7] E.F. Moore and C.E. Shannon. Reliable circuits using less reliable relays. *J. Franklin Institute*, 262:191-208 and 281-297, Sept/Oc. 1956.

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

[9] A. Avizienis. Design of fault-tolerant computers. In *Proc. 1967 Fall Joint Computer Conf., AFIPS Conf. Proc. Vol. 31,* pages 733-743, 1967.

[10] W.G.Bouricius, W.C. Carter, and P.R. Schneider. Reliability modeling techniques for self-repairing computer systems. In *Proceedings of 24th National Conference of ACM,* pages 295-309,1969.

[11] B. Randell. Operating systems: The problems of performance and reliability. In *Proc. IFIP Congress, Vol 1*, pages 281-290. North-Holland, 1971

[12] B. Randell. System structure for software fault tolerance. *IEEE Transctions on Software Engineering,* SE-1:1220-232, 1975.

[13] A. Avizienis and L. Chen. On the implementation of N-version programming for software fault tolerance during execution. In *Proc. IEEE COMPSAC 77*, pages 149-155, November 1977.

[14] Special session. Fundamental concepts of fault tolerance. In *Digest of FTCS-12*, pages 3-38, June 1982.

[15] J.C. Laprie. Dependable computing and fault tolerance: concepts and terminology. In *Digest of FTCS-15*, pages 2-11, June 1985.

[16] J.C. Laprie, editor. *Dependability: Basic Concepts and Terminology*. Springer-Verlag, 1992.

[17] R. Turn and J. Habibi. On the interactions of security and fault tolerance. In *Proc. 9th National Computer Security Conf.*, pages 138-142, September 1986.

[18] J.E. Dobson and B. Randell. Building reliable secure computing systems out of unreliable insecure compnents. In *Proc. of the 1986 IEEE Symp. Security and Privacy*, pages 187-193, April 1986.

[19] J.M. Fray, Y. Deswarte, D. Powell. Intrusion tolerance using fine-grain fragmentation-scattering. In *Proc. 1986 IEEE Symp. on Security and Privacy,* Oakland, Apris 1986, pages 194-201.

[20] M.K. Joseph. Towards the elimination of the effects of malicious logic: Fault tolerance approaches. In *Proc. 10th National Computer Security Conf.*, pages 238-244, September 1987.

[21] M.K. Joseph and A. Avizienis. A fault tolerance approach to computer viruses. In *Proc. of the 1988 IEEE Symposium on Security and Privacy,* pages 52-58, April 1988.

[22] A. Avizienis, J.-C. Laprie, and B. Randell. Dependability of computer systems: Fundamental concepts, terminology, and examples. Technical report, LAAS-CNRS, October 2000.

[23] C.E. Landwehr et al. A taxonomy of computer program security flaws. *ACM Computing Surveys,* 26(3):211-254, September 1994.

[24] Commission Of The European Communities. *Information Technology Security Evaluation Criteria.* 1991.

[25] H.F. Lipson. Survivability — A new security paradigm for protecting highly distributed mission-critical systems. Collection of transparencies, 38th meeting of IFIP WG 10.4, Kerhonson, New York, June 28-July 2, 2000, pp. 63-89. Available from LAAS-CNRS.