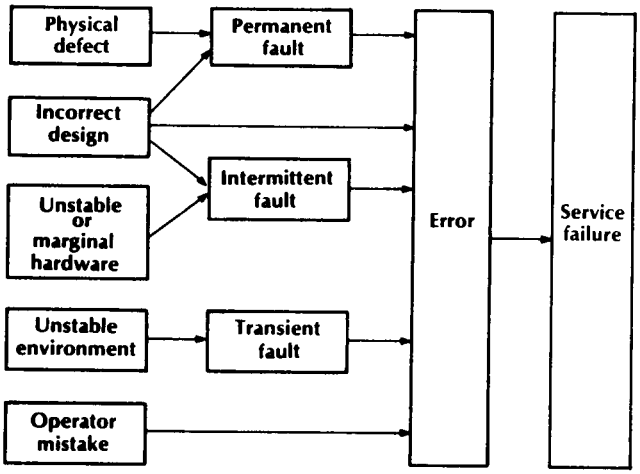
Reliability

The failure of computerized controls in some applications can have catastrophic results. This lecture describes how faults can cause errors which in turn can lead to failures.
After this lecture you should be able to differentiate between availability and reliability. You should also be a able to read a description of a system failure and identify and classify the fault, identify the resulting error and describe strategies for reducing the likelihood of the faults.

Introduction

Computer-controlled systems are used in many applications where the failure of the control system can have serious consequences. Such applications are often found in the fields of transportation (aircraft, trains) , communication (data and telephone networks), weapon systems (radar, missiles), and health care (therapeutic and monitoring equipment).

The purpose of this lecture is to increase awareness of the consequences of system failure. The analysis and design of highly-reliable computer control systems is beyond the scope of this course.



Terminology

Availability is the fraction of time that a system operates correctly. For example, the availability requirement for a telephone switch may be specified as 99.99999% (unavailable for less than 5 minutes per year). Availability is normally specified only for systems that are designed to be repaired.

Reliability is the probability that the device or system will operate correctly in a specified situation. For example, an air traffic control radar may be required to detect a certain size of plane with a probability of 99.999%. A manufacturer may specify the reliability of a system over a given time. For example, the manufacturer of a disk drive may guarantee that the probability of the disk failing in the first 1000 hours of operation is less than 1%.

The following diagram (from Siewiorek and Swarz, 1992) attempts to classify the relationship between faults, errors and failures.

A *fault* is the incorrect operation of part of the system (e.g. temperature sensor failure). The *error* is the effect of that fault (e.g. heat source left on continuously). And a *failure* is the failure of the system to operate as desired (e.g. a fire or damage).

Causes of Failures

Siewiorek and Swarz (1992)

These authors classify causes of failure in two dimensions: the *physical location* of the failure and the *time* during the lifetime of the system when the failure occurs.

The physical location of the failure might range from a component (the lowest level) through to the interconnection of systems (the highest level). In between are the circuit level, and the software level where failures can happen. There are different statistical methods for estimating availability and reliability at each level.

The time at which the failure can be caused can range from the design stage (the earliest time)

through to operation (the latest time). Failures can also be introduced during the prototyping, manufacturing, and test times.

Neumann (1995)

It's also possible to classify most faults into a number of common categories. For example, among the causes listed in (Neumann 1995) are:

- System Analysis - incorrect assumptions or poor models used in the original system design
- Requirements Definition - an important requirement left out or incorrectly specified
- Design Flaws - errors in the design
- Implementation Errors - includes faulty construction, or buggy software
- System use - operator errors
- Hardware malfunction - hardware failures in the field
- Environmental problems - heat, flooding, etc
- Evolution and Maintenance - poorly-tested upgrades or faulty maintenance

Failure-Prevention Strategies

Having identified possible causes of failures, we need to develop strategies to minimize their impacts. Different approaches are used to reduce hardware and software failures. Table **??** identifies some of the techniques.

Hardware

Figure **??** summarizes the various strategies used to reduce the likelihood of hardware failures.

Fault Avoidance includes: conservative design (operation in a benign (low-temperature, clean power, etc) environment, use of reliable components, reduced complexity, etc).

Fault Detection includes: performing the same operation twice and comparing the results, using error-detecting codes such as parity or CRCs, using timers to detect hardware (or software) that is not performing as expected, and detecting accesses to protected or non-existent memory.

Masking Redundancy includes redundant circuitry that hides failures. This can include N-module redundancy (NMR) with voting to select the output that is most likely correct, error-correcting coding and logic design that uses additional gates to correct for partial failures.

Dynamic Redundancy includes replacing failed circuitry with backup circuits. This can include detecting failures and selecting one of N modules, and designing for reduced performance in the case of some failures.

Software

The above strategies can also be used to minimize software failures.

Fault avoidance includes employing good software engineering principles such as modularity, data hiding, sanity checks, regression testing, and bug tracking.

Fault detection and recovery can involve using multiple independently-written programs to perform the same task and methods to recover from errors once they have been detected.

References

Computer Related Risks, Peter G Neumann, ACM Press, 1995. This is an easy-to-read book that is full of examples of the risks involved in computer-based systems. Some of the material isn't relevant to computer control systems but it's all interesting.

`RISKS-LIST`, a "FORUM ON RISKS TO THE PUBLIC IN COMPUTERS AND RELATED SYSTEMS" moderated by the author of the above book and available in the Usenet newsgroup `comp.risks`.

Reliable Computer Systems (Second Edition), Daniel P Siewiorek and Robert W Swarz, Digital Press, 1992. Is a more technical treatment of computer reliability issues.

TABLE 3–1 *Classification of reliable techniques*

Hardware Techniques		Software Techniques	
Class	Technique	Class	Technique
Fault avoidance	Environment modification	Fault avoidance (software engineering)	Modularity
	Quality changes		Object-oriented programming
	Component integration level		Capability-based programming
Fault detection	Duplication		Formal proofs
	Error detection codes	Fault detection	Program monitoring
	M-of-*N* codes, parity, checksums, arithmetic codes, cyclic codes	Masking redundancy	Algorithm construction
			Diverse programming
	Self-checking and fail-safe logic	Dynamic redundancy	Forward error recovery
	Watch-dog timers and timeouts		Backward error recovery
	Consistency and capability checks		Retry, checkpointing, journaling, recovery blocks
	Processor monitoring		
Masking redundancy	NMR/voting		
	Error correcting codes		
	Hamming SEC/DED,[1] other codes		
	Masking logic		
	Interwoven logic, coded-state machines		
Dynamic redundancy	Reconfigurable duplication		
	Reconfigurable NMR[2]		
	Backup sparing		
	Graceful degradation		
	Reconfiguration		
	Recovery		

[1] Single-error correction/double-error detection
[2] *N*-modular redundancy

Table 1: Approaches used to reduce hardware and software failures (from Siewiorek and Swarz, 1992).

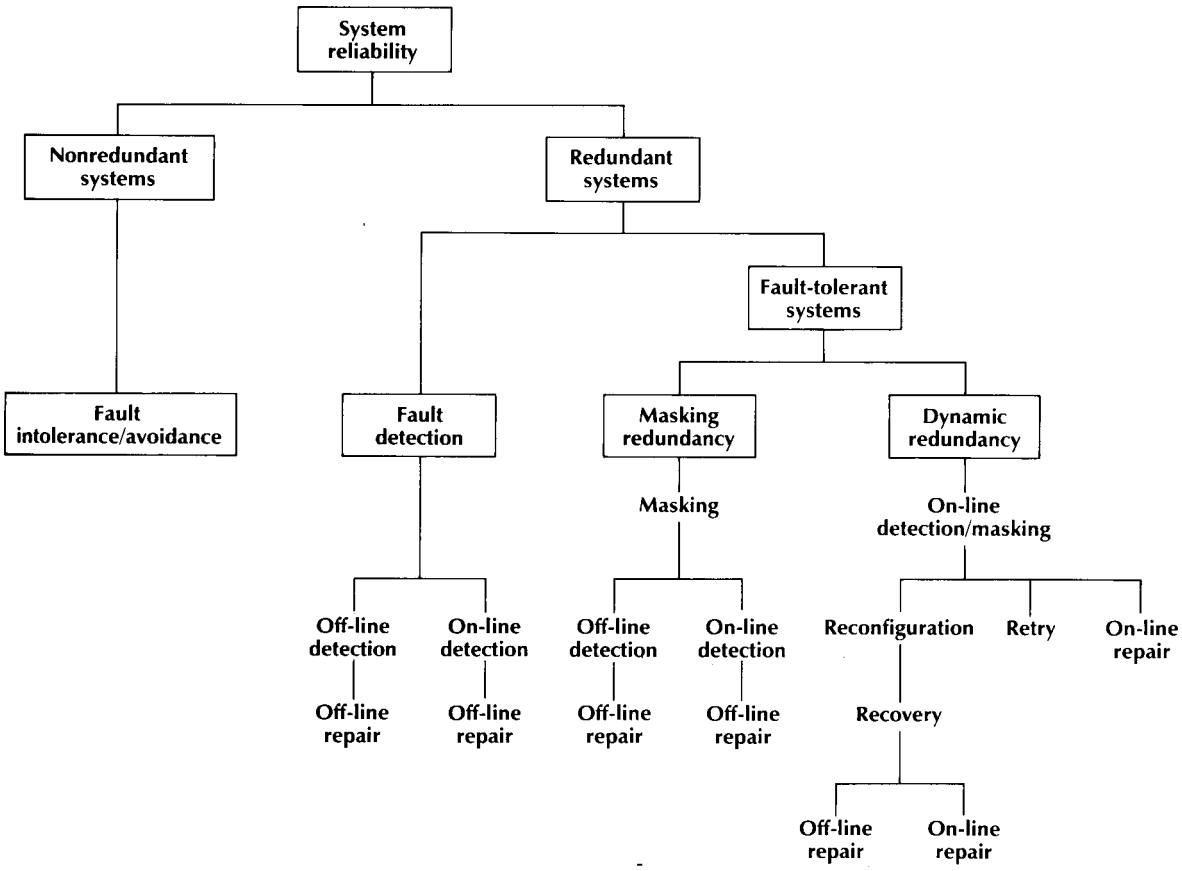


Figure 1: Strategies used to reduce the likelihood of hardware failures (Siewiorek and Swarz).