High Integrity Computing (F23PS2)

High Integrity Software Development (F29HD2)

Faults, Failures and Reliability

Andrew Ireland

Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Overview

- Faults and failures
- Analysis of failure and recovery
- Fault tolerance approaches

Nature of Faults

- Type:
  - Random – hardware component failures
  - Systematic – design faults
- Duration:
  - Permanent – hardware component failures and design faults
  - Transient – appear and disappear after a short time interval *e.g.* alpha particle strikes on semiconductor memory chips – major problem for space probes.
  - Intermittent – appear, disappear then re-appear etc, *e.g.* poor circuit contacts or uninitialized variables.
- Extent:
  - Localized faults
  - Global faults

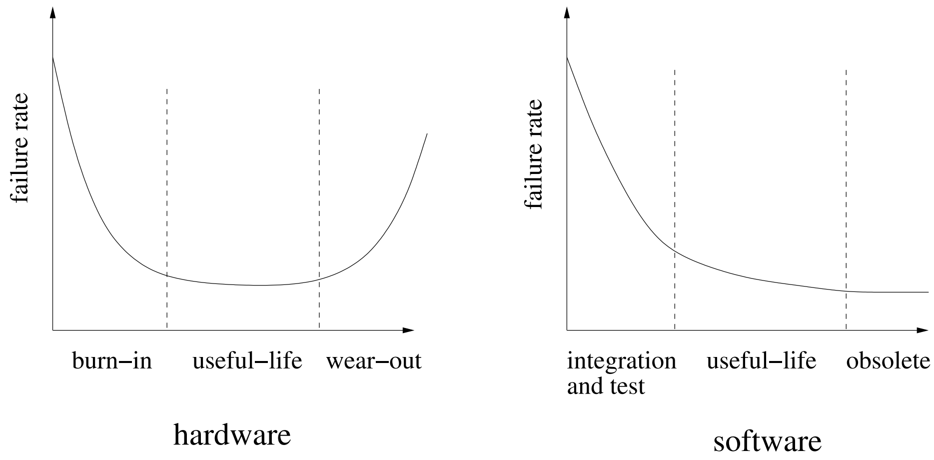
Dealing With Faults

- Fault avoidance & removal: design and implementation techniques that prevent faults from entering a system and remove faults before a system enters service, *i.e.* static and dynamic analysis.
- Fault detection & tolerance: techniques which prevent system faults at run-time leading to system failure.

Faults and Failures

- Faults within a system may or may not lead to system failure.
- Customers are typically concerned with system failures rather than faults.
- Measures of **reliability** are typically of greater interest to customers than measures of faults per lines of code.

Failure Rates



Reliability

- IEEE Definition:

  "The ability of a system or component to perform its required functions under stated conditions for a specified period of time"

- Reliability is typically expressed in terms the **probability** that a system or component will perform its required functions under stated conditions for a specified period of time.

- Reliability is a function of time, *i.e.*

$$R(t) = \frac{n(t)}{N}$$

  where t denotes time, N denotes a set of identical components and $n(t)$ denotes the subset of N that are correctly functioning at time t.

Reliability (More)

- Failure Rate (FR): Average number of times a system is likely to fail in a given period of time, *e.g.* 0.001 failures per hour.

- Mean Time To Failure (MTTF): Expected time that will elapse before the first failure occurs:

$$\text{MTTF} = \frac{1}{\text{FR}}$$

  For example, given FR = 0.001 failures per hour, then:

$$\text{MTTF} = \frac{1}{0.001} = 1000 \text{ hours}$$

Reliability (more)

- Repair Rate (RR): Average number of times a system is likely to be repaired in a given period of time, *e.g.* 0.01 repairs per hour.
- Mean Time To Repair (MTTR): expected time that will elapse during the repair:
$$\mathrm{MTTR} = \frac{1}{\mathrm{RR}}$$
For example, given RR = 0.01 repairs per hour, then:
$$\mathrm{MTTR} = \frac{1}{0.01} = 100 \text{ hours}$$
- Mean Time Between Failures (MTBF): Expected time that will elapse between failures:
$$\mathrm{MTBF} = \mathrm{MTTF} + \mathrm{MTTR}$$

Availability

- Definition:

  The probability that a system will be functioning correctly at any given point in time.

$$\begin{aligned}\text{Availability} &= \frac{\text{Time system is operational}}{\text{Total elapse time}} \\ &= \frac{\mathrm{MTTF}}{\mathrm{MTTF} + \mathrm{MTTR}}\end{aligned}$$

- Note that availability takes account of system maintenance.

Availability (more)

- Assuming failure and repair rates are constant over time then consider:
  - FR = 0.001 failures per hour and
  - RR = 0.01 repairs per hour
- Then it follows:
  - MTTF = 1/0.001 = 1000 hours till first failure and
  - MTTR = 1/0.01 = 100 hours for a repair and
  - Availability = 1000/(1000+100) = 0.909

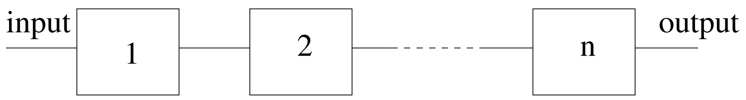
Reliability versus Availability

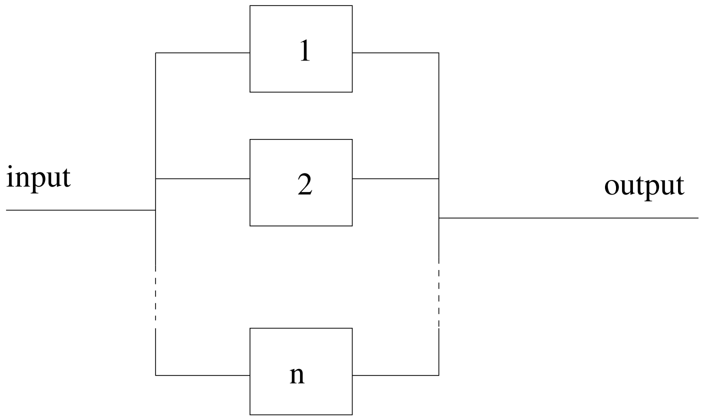
- Both vary with respect to time.
- Reliability relates to a period of time while availability relates to a point in time.
- Availability is a very common requirement of a system, *e.g.* banking systems, telephone networking systems, *etc.*
- Availability often considered more important than reliability if repair time (down-time) can be minimized.
- Availability more important within the context of safety critical systems, *e.g.* a nuclear reactor shutdown software is typically used infrequently but when you need it, your **really** what it to be available!

Reliability versus Safety

- A reliable system does not fail often **but** it provides no guarantee about what happens should it fail. Moreover, a reliable system can put people at risk during normal operation!
- An unreliable system, *i.e.* one that fails often, but which fails in a safe way or recovers quickly can be deemed to be safe, *e.g.* railway signalling systems are fail safe in that on failure a "danger" aspect is displayed.
- Safety may be traded for reliability (availability), *e.g.* In 1984, Bhopal pesticide-producing plant disaster (Union Carbide Ltd): 3,000-16,000 died; 500,000 injured. Safety compromised for the sack of plant reliability.

Series and Parallel Components





Combining Reliabilities (Series)

- If $R_i(t)$ denotes the reliability of component i, then the overall reliability of the system, $R(t)$, is given by:

$$R(t) = R_1(t) \times R_2(t) \times \cdots \times R_n(t)$$

which may be written as:

$$R(t) = \prod_{i=1}^{n} R_i(t)$$

- Note that if the components are identical, *i.e.* all components have reliability $R_k(t)$ then:

$$R(t) = R_k(t)^n$$

Combining Reliabilities (Parallel)

- If $R_i(t)$ denotes the reliability of component i, then $1 - R_i(t)$ is the probability of the component **not** functioning correctly within the time period t.
- In a parallel system the probability of all the parallel components failing within the time t, *i.e.* $Q(t)$, is given by the product of probabilities of each component failing independently:

$$Q(t) = (1 - R_1(t)) \times (1 - R_2(t)) \times \cdots \times (1 - R_n(t))$$

where $Q(t)$ denotes the probability all the components failing within time period t.

Combining Reliabilities (Parallel)

- Given:

$$Q(t) = (1 - R_1(t)) \times (1 - R_2(t)) \times \cdots \times (1 - R_n(t))$$

then it follows that:

$$R(t) = 1 - ((1 - R_1(t)) \times (1 - R_2(t)) \times \cdots \times (1 - R_n(t)))$$

which may be written as:

$$R(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t))$$

- Note that if the components are identical, *i.e.* all components have reliability $R_k(t)$ then:

$$R(t) = 1 - (1 - R_k(t))^n$$

Achieving Fault Tolerance

N-version programming: Given a common specification a number of software teams implement different versions. Operationally the different versions are run in parallel with their outputs compared via a voting scheme. While N-version programming has its benefits, it also has its limitations:

- Specification may be wrong.
- Programmers often make the same mistakes.
- Redundancy management adds complexity.

A NASA study showed that errors associated with the control systems of one of their experimental aircraft were due to defects within the redundancy management system and not the control software.

Achieving Fault Tolerance

Recovery blocks: A local approach to fault tolerance in which sections of code include explicit checks to determine if they have executed successfully. Recovery strategies are:

- Backward recovery: Program execution is returned to a previous state, which hopefully proceeded the event which created the erroneous state, then normal execution is resumed using an alternative section of code.
- Forward recovery: The erroneous part of the current state is repaired and the normal execution is resumed.

Note that N-version programming is just a special case of backward recovery. Backward and forward recovery strategies should be viewed as being complementary rather than alternatives.

Learning Outcomes

- Understand the distinction between faults and failures.
- Understand the differences between the profiles of software and hardware failure rates.
- Understand the notions of reliability and availability, and how they relate to system safety.
- Understand and be able to calculate MTTF, MTTR, MTBF and availability given system (component) failure rates (FR) and repair rates (RR).
- Be able to combine reliability measures (series and parallel).
- Understand basic approaches to fault tolerance.

Recommended Reading

- "Software Reliability Engineering", J.D. Musa , McGraw-Hill, 1998.
- "Handbook of Software Reliability Engineering", M.R. Lyu (Editor), McGraw Hill Text, 1996.