

Reliable Systems



**Idaho State
University**

Computer
Science

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

ROAR



Topics Covered

- Reliability requirements
- Fault-tolerant architectures

Reliability requirements



System reliability requirements

- Functional reliability requirements define system and software functions that avoid, detect or tolerate faults in the software and so ensure that these faults do not lead to system failure.
- Software reliability requirements may also be included to cope with hardware failure or operator error.
- Reliability is a measurable system attribute so non-functional reliability requirements may be specified quantitatively. These define the number of failures that are acceptable during normal use of the system or the time in which the system must be available.



Reliability metrics

- Reliability metrics are units of measurement of system reliability.
- System reliability is measured by counting the number of operational failures and, where appropriate, relating these to the demands made on the system and the time that the system has been operational.
- A long-term measurement program is required to assess the reliability of critical systems.
- Metrics
 - Probability of failure on demand
 - Rate of occurrence of failures/Mean time to failure
 - Availability



Probability of failure on demand

- This is the probability that the system will fail when a service request is made. Useful when demands for service are intermittent and relatively infrequent.
- Appropriate for protection systems where services are demanded occasionally and where there are serious consequence if the service is not delivered.
- Relevant for many safety-critical systems with exception management components
 - Emergency shutdown system in a chemical plant.



Rate of fault occurrence (ROCOF)

- Reflects the rate of occurrence of failure in the system.
- ROCOF of 0.002 means 2 failures are likely in each 1000 operational time units e.g. 2 failures per 1000 hours of operation.
- Relevant for systems where the system has to process a large number of similar requests in a short time
 - Credit card processing system, airline booking system.
- Reciprocal of ROCOF is Mean time to Failure (MTTF)
 - Relevant for systems with long transactions i.e. where system processing takes a long time (e.g. CAD systems). MTTF should be longer than expected transaction length.



Availability

- Measure of the fraction of the time that the system is available for use.
- Takes repair and restart time into account
- Availability of 0.998 means software is available for 998 out of 1000 time units.
- Relevant for non-stop, continuously running systems
 - telephone switching systems, railway signaling systems.



Availability specification

Availability	Explanation
0.9	The system is available for 90% of the time. This means that, in a 24-hour period (1,440 minutes), the system will be unavailable for 144 minutes.
0.99	In a 24-hour period, the system is unavailable for 14.4 minutes.
0.999	The system is unavailable for 84 seconds in a 24-hour period.
0.9999	The system is unavailable for 8.4 seconds in a 24-hour period. Roughly, one minute per week.



Non-functional requirements

- Non-functional reliability requirements are specifications of the required reliability and availability of a system using one of the reliability metrics (POFOD, ROCOF or AVAIL).
- Quantitative reliability and availability specification has been used for many years in safety-critical systems but is uncommon for business critical systems.
- However, as more and more companies demand 24/7 service from their systems, it makes sense for them to be precise about their reliability and availability expectations.



Benefits of reliability specification

- The process of deciding the required level of the reliability helps to clarify what stakeholders really need.
- It provides a basis for assessing when to stop testing a system. You stop when the system has reached its required reliability level.
- It is a means of assessing different design strategies intended to improve the reliability of a system.
- If a regulator has to approve a system (e.g. all systems that are critical to flight safety on an aircraft are regulated), then evidence that a required reliability target has been met is important for system certification.



Specifying reliability requirements

- Specify the availability and reliability requirements for different types of failure. There should be a lower probability of high-cost failures than failures that don't have serious consequences.
- Specify the availability and reliability requirements for different types of system service. Critical system services should have the highest reliability but you may be willing to tolerate more failures in less critical services.
- Think about whether a high level of reliability is really required. Other mechanisms can be used to provide reliable system service.



ATM reliability specification

- Key concerns
 - To ensure that their ATMs carry out customer services as requested and that they properly record customer transactions in the account database.
 - To ensure that these ATM systems are available for use when required.
- Database transaction mechanisms may be used to correct transaction problems so a low-level of ATM reliability is all that is required
- Availability, in this case, is more important than reliability



ATM availability specification

- System services
 - The customer account database service;
 - The individual services provided by an ATM such as 'withdraw cash', 'provide account information', etc.
- The database service is critical as failure of this service means that all of the ATMs in the network are out of action.
- You should specify this to have a high level of availability.
 - Database availability should be around 0.9999, between 7 am and 11pm.
 - This corresponds to a downtime of less than 1 minute per week.



ATM availability specification

- For an individual ATM, the key reliability issues depends on mechanical reliability and the fact that it can run out of cash.
- A lower level of software availability for the ATM software is acceptable.
- The overall availability of the ATM software might therefore be specified as 0.999, which means that a machine might be unavailable for between 1 and 2 minutes each day.



Insulin pump reliability specification

- Probability of failure (POFOD) is the most appropriate metric.
- Transient failures that can be repaired by user actions such as re-calibration of the machine. A relatively low value of POFOD is acceptable (say 0.002) – one failure may occur in every 500 demands.
- Permanent failures require the software to be re-installed by the manufacturer. This should occur no more than once per year. POFOD for this situation should be less than 0.00002.



Functional reliability requirements

- Checking requirements that identify checks to ensure that incorrect data is detected before it leads to a failure.
- Recovery requirements that are geared to help the system recover after a failure has occurred.
- Redundancy requirements that specify redundant features of the system to be included.
- Process requirements for reliability which specify the development process to be used may also be included.



Examples of functional reliability requirements

RR1: A pre-defined range for all operator inputs shall be defined and the system shall check that all operator inputs fall within this pre-defined range. (Checking)

RR2: Copies of the patient database shall be maintained on two separate servers that are not housed in the same building. (Recovery, redundancy)

RR3: N-version programming shall be used to implement the braking control system. (Redundancy)

RR4: The system must be implemented in a safe subset of Ada and checked using static analysis. (Process)

Fault-tolerant architectures



Fault tolerance

- In critical situations, software systems must be fault tolerant.
- Fault tolerance is required where there are high availability requirements or where system failure costs are very high.
- Fault tolerance means that the system can continue in operation in spite of software failure.
- Even if the system has been proved to conform to its specification, it must also be fault tolerant as there may be specification errors or the validation may be incorrect.



Fault-tolerant system architectures

- Fault-tolerant systems architectures are used in situations where fault tolerance is essential. These architectures are generally all based on redundancy and diversity.
- Examples of situations where dependable architectures are used:
 - Flight control systems, where system failure could threaten the safety of passengers
 - Reactor systems where failure of a control system could lead to a chemical or nuclear emergency
 - Telecommunication systems, where there is a need for 24/7 availability.

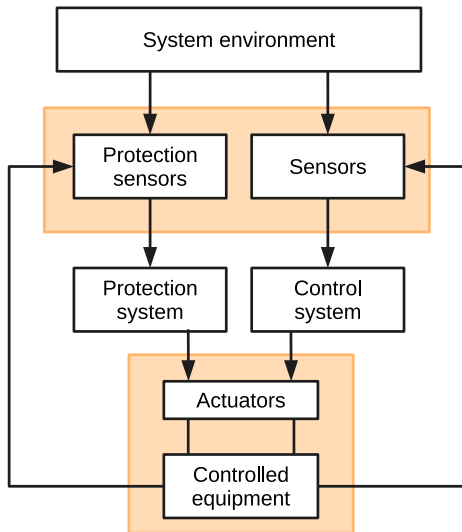


Protection systems

- A specialized system that is associated with some other control system, which can take emergency action if a failure occurs.
 - System to stop a train if it passes a red light
 - System to shut down a reactor if temperature/pressure are too high
- Protection systems independently monitor the controlled system and the environment.
- If a problem is detected, it issues commands to take emergency action to shut down the system and avoid a catastrophe.



Protection system architecture





Protection system functionality

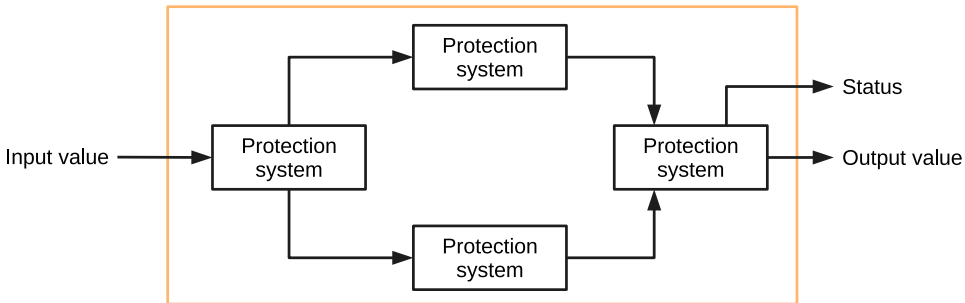
- Protection systems are redundant because they include monitoring and control capabilities that replicate those in the control software.
- Protection systems should be diverse and use different technology from the control software.
- They are simpler than the control system so more effort can be expended in validation and dependability assurance.
- Aim is to ensure that there is a low probability of failure on demand for the protection system.



Self-monitoring architectures

- Multi-channel architectures where the system monitors its own operations and takes action if inconsistencies are detected.
- The same computation is carried out on each channel and the results are compared. If the results are identical and are produced at the same time, then it is assumed that the system is operating correctly.
- If the results are different, then a failure is assumed and a failure exception is raised.

Self-monitoring architecture



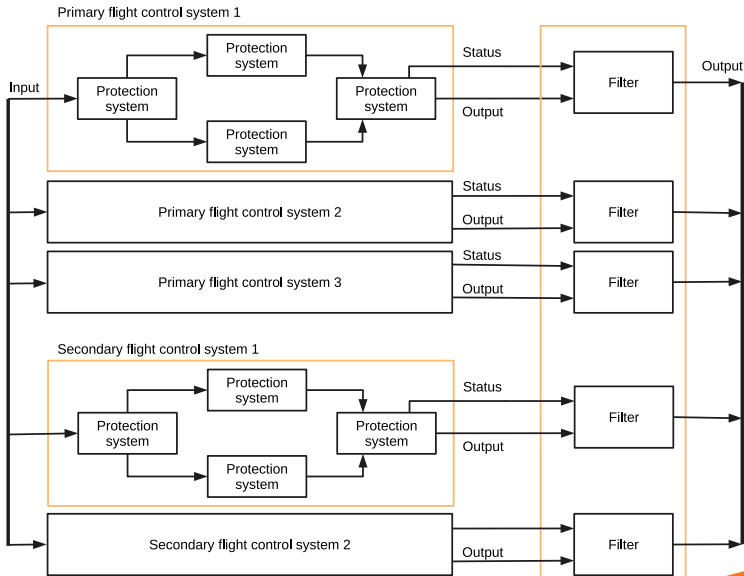


Self-monitoring systems

- Hardware in each channel has to be diverse so that common mode hardware failure will not lead to each channel producing the same results.
- Software in each channel must also be diverse, otherwise the same software error would affect each channel.
- If high-availability is required, you may use several self-checking systems in parallel.
 - This is the approach used in the Airbus family of aircraft for their flight control systems.



Airbus flight control system





Airbus architecture discussion

- The Airbus FCS has 5 separate computers, any one of which can run the control software.
- Extensive use has been made of diversity
 - Primary systems use a different processor from the secondary systems.
 - Primary and secondary systems use chipsets from different manufacturers.
 - Software in secondary systems is less complex than in primary system – provides only critical functionality.
 - Software in each channel is developed in different programming languages by different teams.
 - Different programming languages used in primary and secondary systems.



N-version programming

- Multiple versions of a software system carry out computations at the same time. There should be an odd number of computers involved, typically 3.
- The results are compared using a voting system and the majority result is taken to be the correct result.
- Approach derived from the notion of triple-modular redundancy, as used in hardware systems.

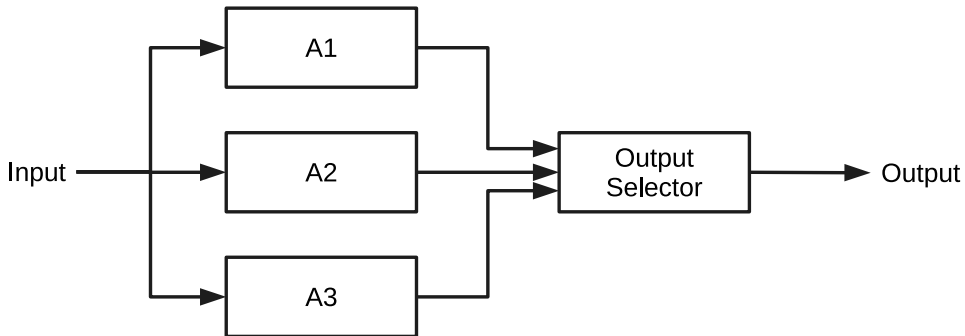


Hardware fault tolerance

- Depends on triple-modular redundancy (TMR).
- There are three replicated identical components that receive the same input and whose outputs are compared.
- If one output is different, it is ignored and component failure is assumed.
- Based on most faults resulting from component failures rather than design faults and a low probability of simultaneous component failure.

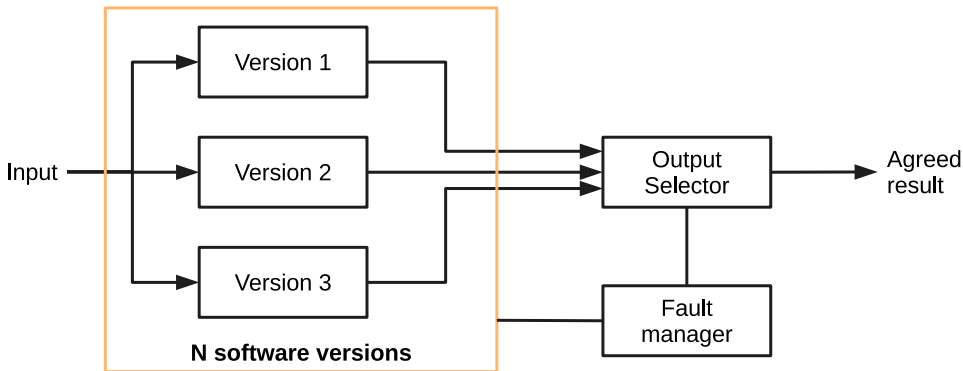


Triple modular redundancy





N-version programming





N-version programming

- The different system versions are designed and implemented by different teams. It is assumed that there is a low probability that they will make the same mistakes. The algorithms used should but may not be different.
- There is some empirical evidence that teams commonly misinterpret specifications in the same way and chose the same algorithms in their systems.



Software diversity

- Approaches to software fault tolerance depend on software diversity where it is assumed that different implementations of the same software specification will fail in different ways.
- It is assumed that implementations are (a) independent and (b) do not include common errors.
- Strategies to achieve diversity
 - Different programming languages
 - Different design methods and tools
 - Explicit specification of different algorithms



Problems with design diversity

- Teams are not culturally diverse so they tend to tackle problems in the same way.
- Characteristic errors
 - Different teams make the same mistakes. Some parts of an implementation are more difficult than others so all teams tend to make mistakes in the same place;
 - Specification errors;
 - If there is an error in the specification then this is reflected in all implementations;
 - This can be addressed to some extent by using multiple specification representations.



Specification dependency

- Both approaches to software redundancy are susceptible to specification errors. If the specification is incorrect, the system could fail
- This is also a problem with hardware but software specifications are usually more complex than hardware specifications and harder to validate.
- This has been addressed in some cases by developing separate software specifications from the same user specification.

Improvements in practice

- In principle, if diversity and independence can be achieved, multi-version programming leads to very significant improvements in reliability and availability.
- In practice, observed improvements are much less significant but the approach seems leads to reliability improvements of between 5 and 9 times.
- The key question is whether or not such improvements are worth the considerable extra development costs for multi-version programming.



Key points

- Dependable system architectures are system architectures that are designed for fault tolerance.
- There are a number of architectural styles that support fault tolerance including protection systems, self-monitoring architectures and N-version programming.
- Software diversity is difficult to achieve because it is practically impossible to ensure that each version of the software is truly independent.



Are there any questions?