

Administrative matter

- Homework #5
 - Due Thursday 11/21
- Guest lecture on Tuesday
 - Susan/Jason present their research activities in embedded system design



Characteristics of a RTS

- Large and complex
- Concurrent control of separate system components
- Facilities to interact with special purpose hardware
- Guaranteed response times
- **Extreme reliability**
- Efficient implementation



Reliability and Fault Tolerance

- Goal
 - To understand the factors which affect the reliability of a system and how software design faults can be tolerated.
- Topics
 - Reliability, failure and faults
 - Failure modes
 - Fault prevention and fault tolerance
 - N-Version programming
 - Software dynamic redundancy
 - The recovery block approach to software fault tolerance
 - A comparison between n-version programming and recovery blocks
 - Dynamic redundancy and exceptions
 - Safety, reliability and dependability



CS122A: Embedded System Design, Fall 01

3

Scope

Four sources of faults which can result in system failure:

- Inadequate specification
- Design errors in software
- Processor failure
- Interference on the communication subsystem



CS122A: Embedded System Design, Fall 01

4

Reliability, Failure and Faults

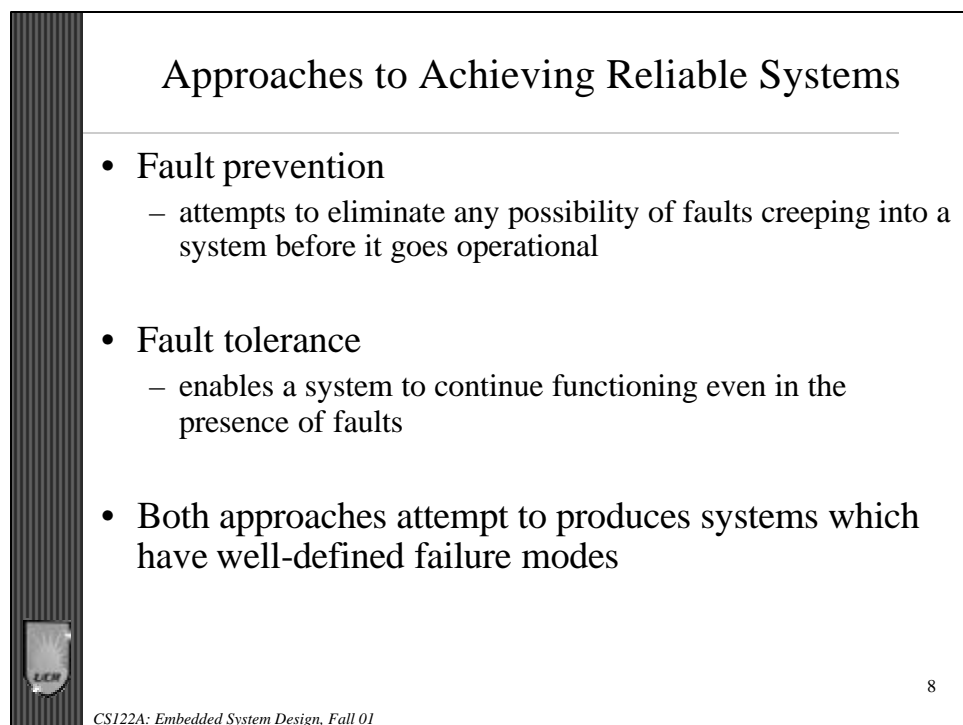
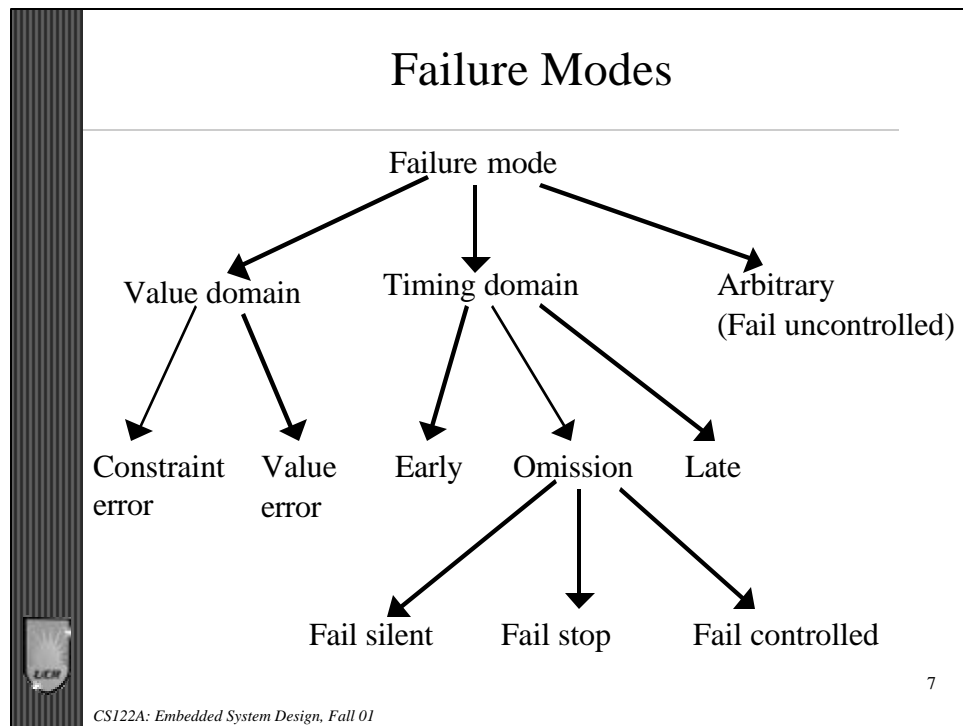
- **Reliability**
 - a measure whether it conforms to some specification of its behaviour
- **Failure**
 - a system deviates from that which is specified for it
 - result from unexpected problems internal to the system
 - eventually manifest themselves in the system's external behaviour
- **Faults**
 - mechanical or algorithmic cause for errors
- **Systems are composed of components**
 - > failure -> fault -> error -> failure -> fault -> error ->
failure -> fault -> error ->



Fault Types

- **Transient fault**
 - starts at a particular time, remains for some period, and then disappears
 - E.g. hardware components which react to radioactivity
 - Many faults in communication systems are transient
- **Permanent faults**
 - remain in the system until they are repaired
 - e.g., a broken wire or a software design error.
- **Intermittent faults**
 - transient faults that occur from time to time
 - E.g. a hardware component that is heat sensitive,
 - it works for a time, stops working, cools down and then starts to work again





Fault Prevention

- Two stages:
 - fault avoidance
 - fault removal
- Fault avoidance limits faults during system construction by:
 - use of the most reliable components
 - use of thoroughly-refined techniques for interconnection
 - packaging the hardware to screen out expected forms of interference.
 - rigorous, if not formal, specification of requirements
 - use of proven design methodologies
 - use of languages with facilities for data abstraction and modularity
 - use of software engineering environments to help manipulate software



CS122A: Embedded System Design, Fall 01

9

Fault Removal

- In spite of fault avoidance, design errors in both hardware and software components will exist
- Fault removal:
 - procedures for finding and removing the causes of errors
 - e.g. design reviews, program verification
- System testing can never be exhaustive and remove all faults
 - A test can only show the presence of faults, not their absence
 - It is sometimes impossible to test under realistic conditions
 - It is difficult to guarantee that the simulation is accurate



CS122A: Embedded System Design, Fall 01

10

Failure of Fault Prevention Approach

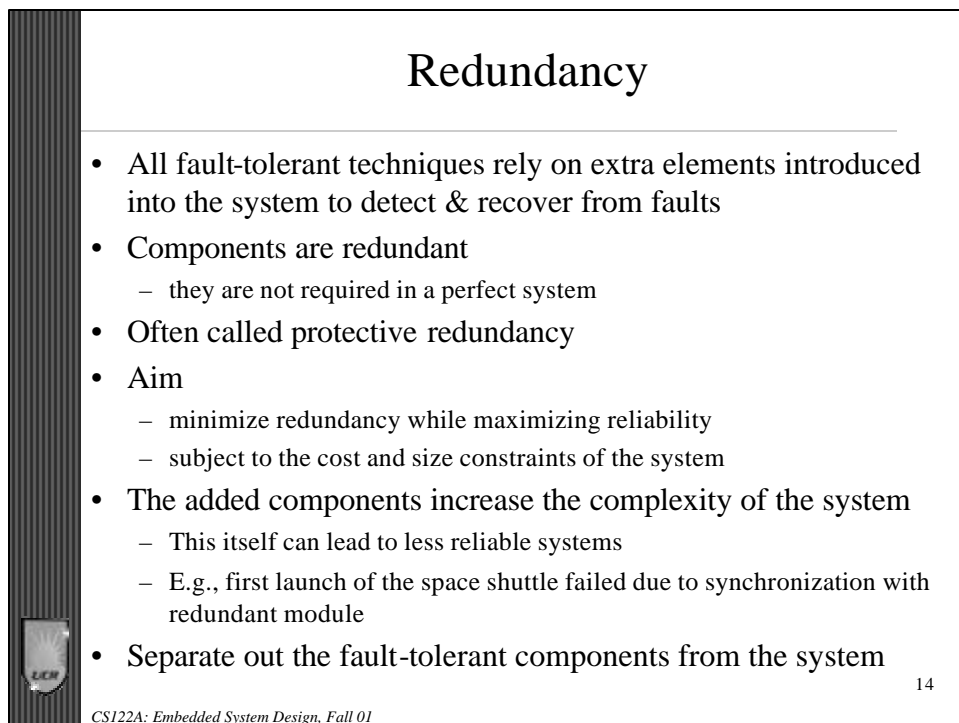
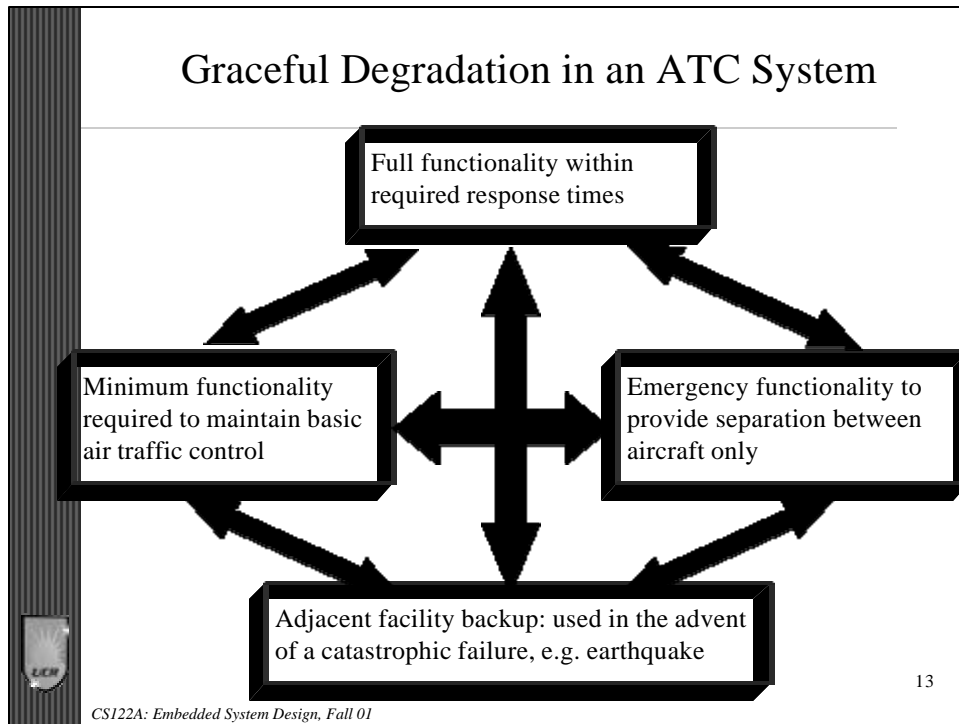
- **Fault will happen!!!**
 - In spite of all the testing and verification techniques
- **The fault prevention approach will be inadequate when**
 - the frequency or duration of repair times are unacceptable
 - the system is inaccessible for maintenance and repair activities
- **An extreme example of the latter**
 - crewless spacecraft Voyager
- **Alternative is **Fault Tolerance****



Levels of Fault Tolerance

- **Full Fault Tolerance**
 - the system continues to operate in the presence of faults
 - with no significant loss of functionality or performance
 - for a limited period
- **Graceful Degradation (fail soft)**
 - the system continues to operate in the presence of errors
 - accepting a partial degradation of functionality or performance
 - for a limited period for recovery or repair
- **Fail Safe —**
 - the system maintains its integrity while accepting a temporary halt
- **The level of fault tolerance required depends on the application**
- **Most safety critical systems require full fault tolerance,**
 - however in practice many settle for graceful degradation






Hardware Fault Tolerance

- Two types of redundancy :
 - static (or masking)
 - dynamic
- Static
 - redundant components are used inside system to hide the effects of faults;
 - e.g. Triple Modular Redundancy
- TMR
 - 3 identical subcomponents and majority voting circuits;
 - the outputs are compared
 - if one differs from the other two that output is masked out
- Assumes the fault is not common
 - Not design error
 - transient or due to component deterioration
- To mask faults from more than one component requires NMR

15




CS122A: Embedded System Design, Fall 01

Hardware Fault Tolerance (Cont.)

- Dynamic redundancy
 - supplied inside a component which indicates that the output is in error
 - provides an error detection facility
 - recovery must be provided by another component
 - E.g. communications checksums and memory parity bits

16



CS122A: Embedded System Design, Fall 01

Software Fault Tolerance

- Used for detecting design errors
- Static
 - N-Version programming
- Dynamic
 - Detection and Recovery
 - Recovery blocks
 - backward error recovery
 - Exceptions
 - forward error recovery



CS122A: Embedded System Design, Fall 01

17

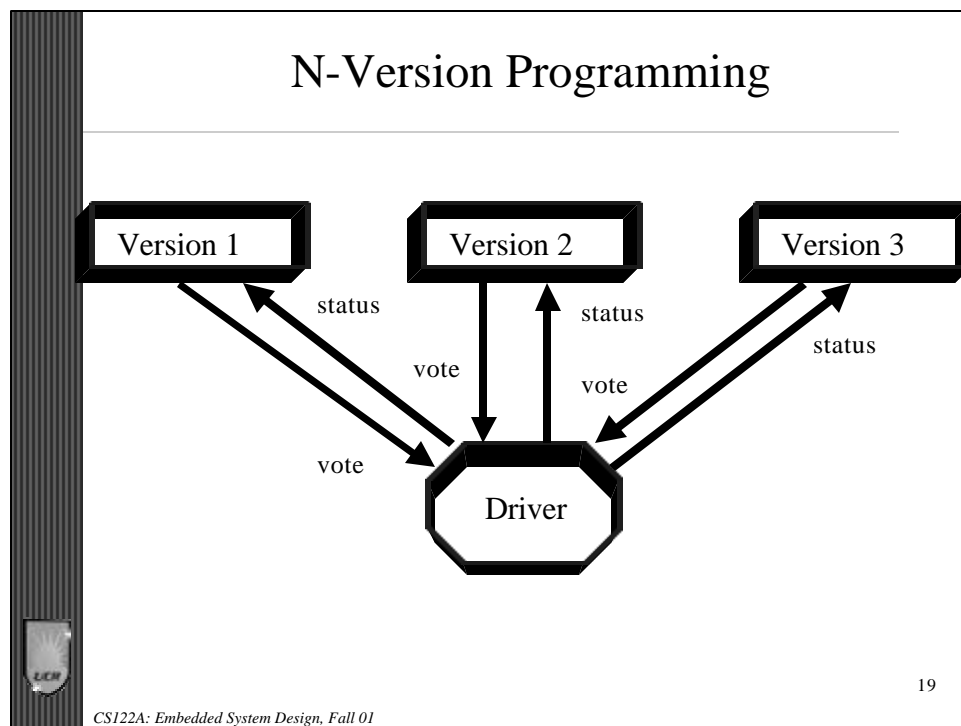
N-Version Programming

- Design diversity
- The independent generation of N ($N > 2$) functionally equivalent programs from the same initial specification
- No interactions between groups
- The programs execute concurrently with the same inputs
- Their results are compared by a driver process
- The results (VOTES) should be identical
- if different
 - the consensus result is taken to be correct
 - assuming there is one



CS122A: Embedded System Design, Fall 01

18

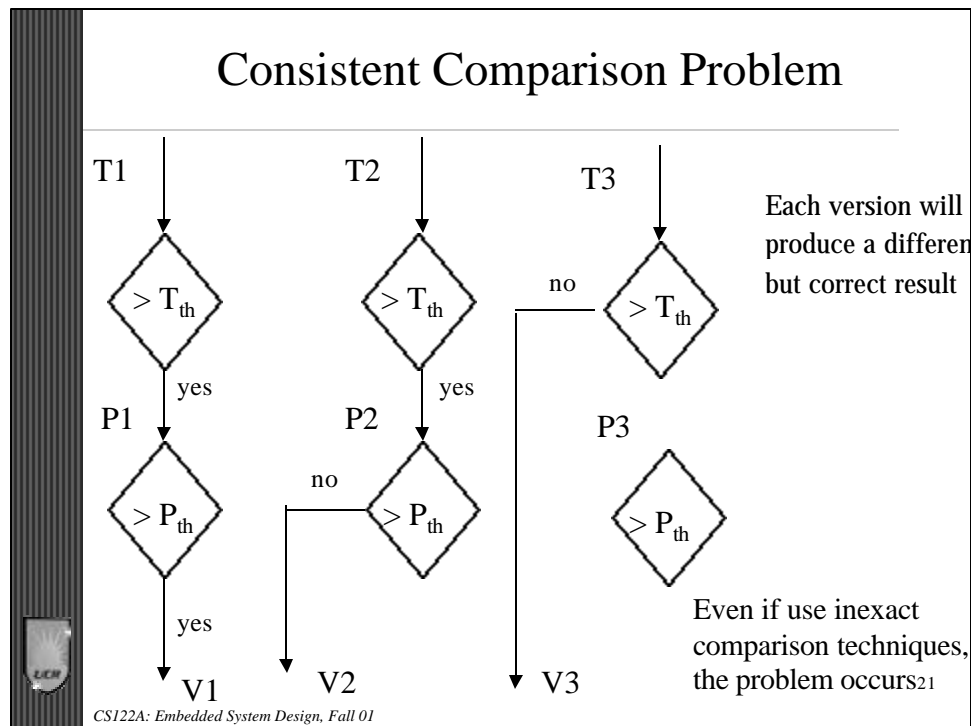


Vote Comparison

- To what extent can votes be compared?
- Text or integer arithmetic will produce identical results
- Real numbers => different values
- More than one result may be correct
 - E.g. Square root
- Need inexact voting techniques

20

CS122A: Embedded System Design, Fall 01



N-version programming depends on

- Initial specification
 - Most software faults stem from inadequate specification
 - A specification error will manifest itself in all N versions
- Independence of effort
 - Experiments produce conflicting results
 - Where part of a specification is complex,
 - this leads to a lack of understanding of the requirements.
- Adequate budget
 - The predominant cost is software
 - A 3-version system will
 - triple the budget requirement
 - cause problems of maintenance.
 - Would a more reliable system be produced if the resources were instead used to produce a single version?

22

CS122A: Embedded System Design, Fall 01

Software Dynamic Redundancy

Four phases

- error detection
 - no fault tolerance scheme can be used until the associated error is detected
- damage confinement and assessment
 - to what extent has the system been corrupted?
 - The delay between a fault occurring and the detection of the error means erroneous information could have spread throughout the system
- error recovery
 - techniques should aim to transform the corrupted system into a state from which it can continue its normal operation
- fault treatment and continued service
 - an error is a symptom of a fault
 - although damage repaired, the fault may still exist

23

CS122A: Embedded System Design, Fall 01

Error Detection

- Environmental detection
 - hardware
 - e.g. illegal instruction
 - O.S/RTS
 - null pointer

24

CS122A: Embedded System Design, Fall 01

Error Detection (Cont.)

- Application detection
 - Replication checks
 - N-version programming
 - Timing checks
 - Watchdog timer
 - Reversal checks
 - Square root function: square the result to check
 - Coding checks
 - checksum
 - Reasonableness checks
 - Range checking
 - Structural checks
 - counts
 - Dynamic reasonableness check
 - causality

25

CS122A: Embedded System Design, Fall 01

Damage Confinement and Assessment

- Concerned with structuring the system so as to minimise the damage caused by a faulty component
 - Firewalling
- Modular decomposition
 - provides static damage confinement
 - allows data to flow through well-define pathways
- Atomic actions
 - E.g. if started, must be able to complete
 - E.g. only one process access memory at a time
 - provides dynamic damage confinement
 - they are used to move the system from one consistent state to another

26

CS122A: Embedded System Design, Fall 01

Error Recovery

- Two approaches:
 - forward
 - backward
- Forward error recovery
 - continues from an erroneous state by making selective corrections
 - This includes making safe the controlled environment
 - which may be hazardous or damaged because of the failure
 - system specific
 - depends on accurate predictions of the location and cause of errors
 - Examples: redundant pointers in data structures



CS122A: Embedded System Design, Fall 01

27

Backward Error Recovery (BER)

- Relies on restoring the system to a previous safe state
 - Then executing an alternative section of the program
- This has the same functionality but
 - uses a different algorithm (c.f. N-Version Programming)
- Recovery point
 - The point to which a process is restored
- checkpointing (saving appropriate system state)
 - the act of establishing it is termed
- Advantage
 - the erroneous state is cleared
 - it does not rely on finding the location or cause of the fault
 - recover from unanticipated faults including design errors
- Disadvantage
 - Cannot undo errors in the environment



CS122A: Embedded System Design, Fall 01

28

The Domino Effect

- With concurrent processes that interact with each other, BER is more complex

CS122A: Embedded System Design, Fall 01

Fault Treatment and Continued Service

- ER returned the system to an error-free state;
 - however, the error may recur
 - the final phase of F.T. is to eradicate the fault from the system
- The automatic treatment of faults is difficult
- Some systems assume all faults are transient
 - others that error recovery techniques can cope with recurring faults
- Fault treatment can be divided into 2 stages:
 - fault location
 - system repair
- Error detection techniques can help to trace the fault
 - The component can be replaced
- In non-stop applications
 - it will be necessary to modify the program while it is executing!

30

CS122A: Embedded System Design, Fall 01

The Recovery Block approach to FT

- At the entrance to a block is an automatic recovery point
- At the exit an acceptance test
- Acceptance test
 - used to test that the system is in acceptable state after execution
 - If the acceptance test fails,
 - the program is restored to the recovery point at the beginning of the block
 - an alternative module is executed
- If the alternative module also fails the acceptance test
 - the program is restored to the recovery point
 - yet another module is executed, and so on
- If all modules fail then the block fails



CS122A: Embedded System Design, Fall 01

31

Recovery Block Syntax

```

ensure <acceptance test>
by
    <primary module>
else by
    <alternative module>
else by
    <alternative module>
    ...
else by
    <alternative module>
else error
  
```

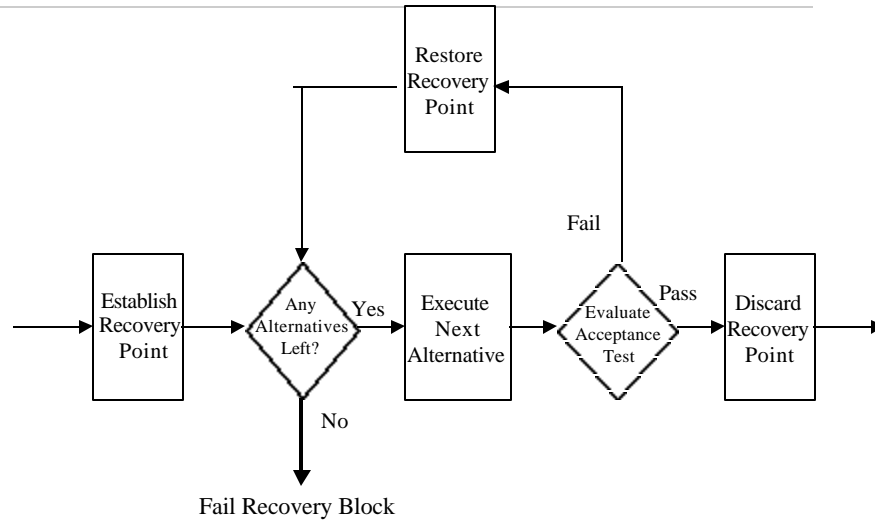
- Recovery blocks can be nested
- If all alternatives in a nested recovery block fail,
 - the outer level recovery point will be restored
 - an alternative module to that block executed



CS122A: Embedded System Design, Fall 01

32

Recovery Block Mechanism



33

CS122A: Embedded System Design, Fall 01

Example: Solution to Differential Equation

```

ensure Rounding_err_has_acceptable_tolerance
by
  Explicit Kutta Method
else by
  Implicit Kutta Method
else error
  
```

- **Explicit Kutta Method**
 - fast but inaccurate when equations are stiff
- **Implicit Kutta Method**
 - more expensive but can deal with stiff equations
- The above will cope with all equations
- It will also potentially tolerate design errors in the Explicit Kutta Method if the acceptance test is flexible enough

34

CS122A: Embedded System Design, Fall 01

Nested Recovery Blocks

```
ensure rounding_err_has_acceptable_tolerance
by
  ensure sensible_value
  by
    Explicit Kutta Method
  else by
    Predictor-Corrector K-step Method
  else error
else by
  ensure sensible_value
  by
    Implicit Kutta Method
  else by
    Variable Order K-Step Method
  else error
else error
```

35

CS122A: Embedded System Design, Fall 01

The Acceptance Test

- Acceptance test
 - provides the error detection mechanism
 - enables the redundancy in the system
- There is a trade-off between
 - providing comprehensive acceptance tests and
 - keeping overhead to a minimum, so fault-free execution is not affected
- Note that the term used is acceptance not correctness
 - this allows a component to provide a degraded service
- Error detection techniques can be used

36

CS122A: Embedded System Design, Fall 01

N-Version Programming vs Recovery Blocks

- Static (NV) versus dynamic redundancy (RB)
- Design overheads
 - both require alternative algorithms
 - NV requires driver, RB requires acceptance test
- Runtime overheads
 - NV requires $N * \text{resources}$, RB requires establishing recovery points
- Diversity of design
 - both susceptible to errors in initial requirements
- Error detection
 - vote comparison (NV) versus acceptance test(RB)
- Atomicity
 - NV vote before it outputs to the environment
 - RB must be structure to only output after passing of an acceptance test

37

CS122A: Embedded System Design, Fall 01

Dynamic Redundancy and Exceptions

- An exception can be defined as the occurrence of an error
- raising (or signally or throwing)
 - Bringing an exception to the attention of the invoker
- handling (or catching)
 - The invoker's response
- Exception handling is a forward error recovery
 - there is no roll back to a previous state
 - control is passed to the handler
 - However, the handler can be used to provide backward ER

38

CS122A: Embedded System Design, Fall 01

Exceptions

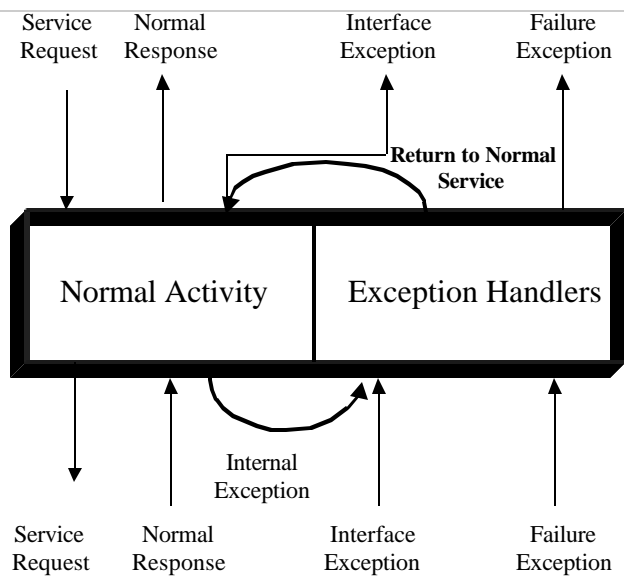
Exception handling can be used to:

- cope with abnormal conditions in the environment
- enable program design faults to be tolerated
- provide general-purpose error-detection and recovery



39

Ideal Fault-Tolerant Component



40

Safety and Reliability

- Safety
 - freedom from those conditions that can
 - cause death, injury, occupational illness, damage to (or loss of) equipment (or property), or environmental harm
- By this definition, most systems which have an element of risk associated with their use as unsafe
- Reliability:
 - a measure of the success with which a system conforms to specification



Safety Vs. Reliability

- Example
 - Measures which increase the likelihood of a weapon firing when required may well increase the possibility of its accidental detonation
- In many ways, the only safe airplane is one that never takes off however, it is not very reliable.



