CSE 598D: Software Fault Tolerance

Instructor: Mahmut Kandemir
Email: kandemir@cse.psu.edu
210 Thomas, TR 2:30-3:45
Textbook

- No textbook

Useful references

- Software fault tolerance techniques and implementation

- Software Reliability Engineering
Topics to be covered

1. Definitions of Software Reliability Concepts
2. Redundancy Structuring for Fault Tolerance
3. Reliability Oriented Design Methods and Programming Techniques
4. Design Diverse Software Fault Tolerance Techniques
5. Data Diverse Software Fault Tolerance Techniques
6. Adjudicating the Results
7. Software Approaches to Hardware Reliability
Why software fault tolerance?

Computer-based systems have increased dramatically in scope, complexity, and pervasiveness.

Safe and reliable software operation is a significant requirement for many systems:

- Aircraft, medical devices, nuclear safety, electronic banking and commerce, automobiles, etc, ...

Consequences of these systems failing can range from mildly annoying to catastrophic.

Software assumes more of the responsibility for providing functionality in these systems.
Why are there errors in software?

- The current state-of-the-practice is such that fewer errors are introduced, but unfortunately not all errors are prevented.

- Even if the best people, practices, and tools are used, it would be very risky to assume the software developed is error-free.

- There may also be cases in which an error, found late in the system’s life cycle and perhaps prohibitively expensive to repair, is knowingly allowed to remain in the system.
Software-related accidents and incidents

- Problems in the backup tracking software delayed the launch of Atlantis for three days.

- AT&T system suffered a nine-hour United States wide blockade due to a flaw in recover-recognition software.

- During Gulf War, the Patriot system missed a missile due to a clock shift caused by the software’s use of two different and unequal representations (24-bit and 48-bit) of the value 0.1.
Unique challenges compared to hardware faults

- Hardware faults are primarily physical faults, which can be characterized and predicted over time.
- Software has only logical faults, which are difficult to visualize, classify, detect, and correct.
- Software faults may be traced to incorrect requirements or to the implementation not satisfying the requirements.
- Changes in operational usage or incorrect modifications may introduce new faults.
- Redundancy is not enough to protect against these faults.
Dependability concept classification

- Dependability
  - Impairments
    - Faults
    - Errors
    - Failures
  - Means
    - Construction
    - Fault avoidance
    - Fault tolerance
    - Fault removal
    - Fault forecasting
  - Attributes
    - Availability
    - Reliability
    - Safety
    - Confidentiality
    - Integrity
    - Maintainability
Dependability concept classification

- **Impairments**: those things that stand in the way of dependability
- **Means**: the various technical means to achieve dependable software
- The **attributes** of dependability enable the properties of dependability and provide a way to assess achievement of those properties
Impairments

Dependability

Means

Attributes

Faults

Errors

Failures

Construction

Validation

Availablility

Reliability

Safety

Confidentiality

Integrity

Maintainability

Fault avoidance

Fault tolerance

Fault removal

Fault forecasting
A **fault** is the identified or hypothesized cause of an error

Sometimes called a “bug”

It can be viewed as simply the “consequence of a failure of some other system that has delivered or is now delivering a service to the given system”

An active fault is one that produces an error
An error is part of the system state that is liable to lead to a failure.

It can be unrecognized as an error (latent) or detected.

An error may propagate, i.e., produce other errors.

Faults are known to be present when errors are detected.

An error is a manifestation of a fault.
A failure occurs when the service delivered by the system deviates from the specified service, otherwise termed as an incorrect result.

The expected service is described, typically by a specification or set of requirements.

The cycle: ...failure → fault → error → failure → fault...
Means to achieve dependable software

- Dependability
  - Impairments
    - Faults
    - Errors
    - Failures
  - Means
    - Construction
      - Fault avoidance
      - Fault tolerance
      - Fault removal
      - Fault forecasting
    - Validation
  - Attributes
    - Availability
    - Reliability
    - Safety
    - Confidentiality
    - Integrity
    - Maintainability
Means to achieve dependable software

Two major groups

- **Construction**: those that are employed during the software construction process
- **Validation**: those that contribute to validation of the software after it is developed
Fault avoidance or prevention techniques are dependability enhancing techniques employed during software development to reduce the number of faults introduced during construction.

These techniques may address:

- System requirements specification
- Structured design and programming methods
- Formal methods
- Software reuse
System requirement specification

- A system failure may occur due to logic errors incorporated into the requirements.
- This results in software that is written to match the requirements, but the behavior specified in the requirements is not the expected or desired behavior.
- The specification of system requirements is currently an imperfect process at best.
Software requirements specification lies between software engineering and system engineering

- These two disciplines suffer from a lack of communication
- Software engineers tend to work in isolation from the rest of a system’s developers
- Majority of safety problems arise from software requirements errors and not from coding errors

A large part of existing software engineering techniques addresses only the errors that occur when the design and implementation of the requirements do not match the specification
Structured design and programming methods

Structured software design and programming methods have been shown to be effective and are in common use.

Techniques

- Introducing structure to the design
- Decoupling and modularization
- Information hiding (OOP & OOD)

Each of the techniques reduces overall complexity of the software and reduces the introduction of faults into the software.
In formal methods, requirements specifications are developed and maintained using mathematically tractable languages and tools.

Four goals of current formal method studies:

- Executable specifications for systematic and precise evaluation
- Proof mechanisms for software verification and validation
- Development procedures that follow incremental refinement for step-by-step verification
- Every work item, be it a specification or a test case, is subject to mathematical verification for correctness and appropriateness
Mathematical specifications of proofs of software properties tend to be the same size as the program

- Difficult to construct
- Often harder to understand than the program itself
- They can be just as prone to error as the software

Formal methods have not been generally used on large projects
Software reuse

- Software reusability implies a savings in development cost
- It can also increase dependability because software that has been well exercised is less likely to fail
- Object-oriented paradigms and techniques encourage and support software reuse
- Different measures of dependability may not be improved equally by reuse of software
Fault avoidance/prevention summary

- System requirements specification – generally accepted and widely used
- Structured design and programming methods – popular in developing software
- Formal methods – thorough, large overhead, acceptable only in small components – an active research area
- Software reuse – popular and helpful, but simply reusing code does not ensure dependability
- Despite fault prevention efforts, faults are created, so fault removal is needed!
Fault removal

Fault removal techniques are dependability-enhancing techniques employed during software verification and validation.

Improving software dependability by:
- Detecting existing faults, using verification and validation (V&V) methods
- Eliminating the detected faults

Techniques:
- Software testing
- Formal inspection
- Formal design proofs
Software testing

- Software testing is one of the most common fault removal techniques
- Difficulties encountered in testing
  - Prohibitive cost – you cannot test entire software
  - Complexity of exhaustive testing
- Key to efficient testing
  - Maintain an adequate test coverage
  - Derive appropriate test quality measures
  - Minimizing component size and interrelationships maximizes accurate testing
- Additional testing may be performed on components identified as critical to the system
Formal inspection is a rigorous process, accompanied by documentation that focuses on:

- Examining source code to find faults
- Correcting the faults
- Verifying the corrections

A practical and success fault removal technique widely implemented in industry.

Usually performed by small peer groups prior to the testing phase of the life cycle.
Formal design proofs

- **Formal design proofs**: using executable specifications, test cases can be automatically generated to improve the software verification process.
  - Attempts to achieve mathematical proof of correctness of programs.
  - Closely related to formal methods.
- An emerging technique.
  - May be costly and complex.
  - May give the designer a high degree of confidence.
Fault removal summary

Fault removal techniques

- Determine whether the software matches the specified required behavior
- They do not determine whether something has been left out of the requirements

Fault removal is imperfect, so fault forecasting and fault tolerance are needed!
Fault/failure forecasting includes dependability enhancing techniques that are used during the validation of software to estimate the presence of faults and the occurrence and consequence of failures.

Usually focuses on the reliability measure of dependability.

Also known as software reliability measurement.
Fault/failure forecasting (cont.)

It involves

- The formulation of a fault/error/failure relationship
- An understanding of the operational environment
- The establishment of reliability models
- The collection of failure data
- The application of reliability models by tools
- The selection of appropriate models
- The analysis and interpretation of results
- Guidance for management decisions
Two types of activities of fault forecasting

- Reliability estimation
- Reliability prediction
Reliability estimation

- Reliability estimation determines current software reliability by applying statistical inference techniques to failure data obtained during system testing or during system operation.
- Reliability estimation is a snapshot of the reliability that has been achieved to the time of estimation.
Reliability prediction

Reliability prediction determines future software reliability based upon available software metrics and measures.

Different techniques are used depending on the software development stage.
Fault/failure forecasting summary

Fault forecasting techniques determine whether the software matches the specified required behavior

- They reveal whether additional testing, fault tolerance, or other reliability-enhancing measures are needed
- They do not determine whether something has been left out of the requirements

- Missing/incorrect requirements is one of the most fundamental impediments to software dependability
- Fault/failure forecasting can indicate the need for fault tolerance
Fault tolerance techniques provide mechanisms to the software system to prevent system failure from occurring when a fault occurs:

- Reduces the risks of software design faults
- Enables a system to tolerate software faults remaining in the system after its development

Techniques:

- Single version software techniques
- Multiple version software techniques
- Multiple data representation techniques
Single version software environment

- Monitoring techniques
- Atomicity of actions
- Decision verification
- Exception handling
Multiple version software environment

Design diverse techniques are used in this environment.

These techniques utilize functionally equivalent yet independently developed software versions to provide tolerance to software design faults.

Example techniques:
- Recovery blocks (RcB)
- N-version programming (NVP)
- N self-checking programming (NCSP)
Multiple data representation environment

Data diverse techniques are used in this environment.

These techniques utilize different representations of input data to provide tolerance to software design faults.

Example techniques:

- Retry blocks (RtB)
- N-copy programming (NCP)
Software fault tolerance summary

- Fault tolerance techniques are designed to allow a system to tolerate software faults that remain in the system after its development.
- They provide protection against errors in translating the requirements and algorithms into a programming language.
- They do not provide explicit protection against errors in specifying the requirements.
Software fault tolerance process

The fault tolerance process is the set of activities whose goal is to remove errors and their effects from the computational state before a failure occurs.

It consists of:

- Error detection: an erroneous state is identified
- Error diagnosis: the damage caused by the error is assessed and the cause of the error is determined
- Error containment/isolation: further damages are prevented
- Error recovery: the erroneous state is substituted with an error-free state
More concepts

- Types of recovery
- Types of redundancy
Types of error recovery

- Backward recovery
- Forward recovery
Backward recovery

- The most generally applicable recovery technique for software tolerance
- It is usually assumed that the previously saved state is error-free
- The state should checkpointed on stable storage that will not be affected by failure
- The RcB technique and most distributed systems incorporating software fault tolerance employ backward recovery
Alternatives to checkpointing

- Incremental checkpointing
- Audit trail
- Logs
Advantages of backward recovery

- Handles unpredictable errors caused by residual design faults if the errors do not affect the recovery mechanism
- Can be used regardless of the damage sustained by the state
- Provides a general recovery scheme
- The only knowledge required of the errors is that the relevant prior state is error-free
- Particularly suited to recovery of transient faults
Disadvantages of backward recovery

- Requires significant resources to perform checkpointing and recovery
- Implementation of backward recovery often requires that the systems be halted temporarily
- Domino effect may occur
- Additional complications for parallel/distributed environments (?)
After an error occurs, forward recovery attempts to find a new state from which the system can continue operation.

- Can employ error compensation.
- Techniques using forward recovery:
  - NVP
  - NCP
  - DRB (distributed recovery block)

Forward recovery is primarily used when there is no time for backward recovery.

- Finding the new state:
  - Degraded mode of the previous error-free state
  - Error compensation
Error compensation

Error compensation is based on an algorithm that uses redundancy to select or derive the correct answer or an acceptable answer.

If used with self-checking components, then state transformation can be induced by switching from a failed component to a non-failed one executing the same task.

Error compensation may be applied all the time, whether or not an error occurred.

Fault masking
Advantages of forward recovery

- Fairly efficient in terms of the overhead it requires
  - Crucial in real-time applications
- If the fault is an anticipated one, then redundancy and forward recovery can be a useful and timely approach
- Faults involving missed deadlines may be better recovered from using forward recovery than by introducing additional delay in rolling back and recovery
- Provide more efficient solution when characteristics of a fault are well understood
Disadvantages of forward recovery

- Application-specific, i.e., it must be tailored to each situation or program
- Removes only predictable errors
- Requires knowledge of the error
- Cannot aid in recovery if the state is damaged beyond “specification-wise recoverability”
- Depends on the ability to accurately detect the occurrence of a fault, predict potential damage from a fault, and assess the actual damage
Types of redundancy

Redundancy is a key supporting concept for fault tolerance.

Redundancy provides the additional capabilities and resources needed to detect and tolerate faults.

Several forms:
- Hardware
- Software
- Information
- Time
Hardware redundancy

- Includes replicated and supplementary hardware added to the system to support fault tolerance
- The most common use of redundancy
- Redundant or diverse software can reside on redundant hardware to tolerate both hardware and software faults
- **Pure hardware redundancy cannot tolerate software faults**
Software redundancy

- Software redundancy includes additional programs, modules, functions, or objects used to support fault tolerance.
- Software faults overwhelmingly arise from specification and design errors or implementation mistakes.
- Software design and implementation errors cannot be detected by simple replication of identical software units.
- A solution is to introduce diversity into the software replicas.
Software diversity

The goals of increasing diversity in software components are

- To decrease the probability of similar, common-use, coincident, or correlated failures
- To increase the probability that the components fail on disjoint subsets of the input space

When diversity is used, the redundant software components are termed variants, versions, or alternates

The basic approach: start with the same specification and have different programming teams develop the variants independently
Adjudicator

- Adjudicator (or decision mechanism) adjudicates, arbitrates, or otherwise decides on the acceptability of the results obtained by the variants
- Use of diverse software modules requires an adjudicator
- This adjudication module is not replicated and typically does not have an alternate
- It is very important that the adjudicator itself is free from errors
Forms of software redundancy

- All replicas on a single hardware component
- Replicas on multiple hardware components
- The adjudicator on a separate hardware component
- The software that is replicated can range from an entire program to a few lines of code
- Choices to be made are based on available resources and on the specific application
Information or data redundancy

- Information or data redundancy includes the use of information with data and the use of additional forms of data to assist in fault tolerance.

- The addition of data information is typically used for hardware fault tolerance:
  - E.g., error-detecting and error-correcting codes

- Diverse data can be used for tolerating software faults:
  - E.g., data re-expression algorithm (DRA) produces different representations of a module’s input data.
Temporal redundancy

Temporal redundancy involves the use of additional time to perform tasks related to fault tolerance.

It can be used for both hardware and software fault tolerance.

It commonly comprises repeating an execution using the same software and hardware resources involved in the initial failed execution.

Backward recovery schemes typically use a combination of temporal and software redundancy.
Transient faults

- Timing or transient faults arise from the often complex interaction of hardware, software, and the operating system.
  - Difficult to duplicate and diagnose.
  - Also called Heisenbugs.
- Simple replication of redundant software or of the same software can overcome transient faults.
Advantages and disadvantages of temporal redundancy

Advantages

- Simply requires the availability of additional time to re-execute the failed process
- Suitable for applications in which time is readily available, such as human-interactive programs

Disadvantage

- Not suitable for applications with hard real-time constraints
Summary

- Growing need for dependable systems
- Combination of techniques
  - Fault avoidance, fault removal, fault forecasting, fault tolerance
  - Neither forward nor backward recovery is ideal
  - Most fault tolerance techniques based on some form of redundancy
    - software, information, and/or time
    - Selection/combination is situation specific
Means to achieve dependable software

Dependability
- Impairments
  - Faults
  - Errors
  - Failures
- Means
  - Construction
    - Fault avoidance
    - Fault tolerance
  - Validation
    - Fault removal
    - Fault forecasting
- Attributes
  - Availability
  - Reliability
  - Safety
  - Confidentiality
  - Integrity
  - Maintainability
Attributes – Reliability and Availability

Reliability of a component is its ability to function correctly over a specified period of time

\[ R(t) = \Pr(S \text{ is functioning in } [0,t]) \]

Instantaneous or point availability (also called transient availability) of a component is defined to be the probability that system is working at the instant \( t \), regardless of the number of times it may have failed and been repaired in the interval \((0,t)\)

What if the component is not repairable?