Models and techniques to evaluate Software and System reliability
Outline

- Dependability Concepts
- Means to achieve dependable software
- Reliability Measures
- Reliability Models
- Case Study
- Possible Tool usage
- Suggestions
Dependability Concepts
General Definition

- Dependability is a value showing the reliability of a person to others because of his/her integrity, truthfulness, and trustfulness, traits that can encourage someone to depend on him/her.
Dependability as applied to computer systems is defined as the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers.

Dependability can be thought of as being composed of three elements:

- Impairments
- Means
- Attributes
The main characteristics of dependability can be summarized in the form of a tree:

- **Impairments**
  - Faults
  - Errors
  - Failures

- **Means**
  - Construction
  - Validation
  - Fault Prevention
  - Fault Tolerance
  - Fault Removal
  - Fault Forecasting

- **Attributes**
  - Security
    - Availability
    - Integrity
    - Confidentiality
  - Reliability
  - Safety
  - Maintainability
Impairments, Means and Attributes

- Impairments
  - Things that can affect Dependability
- Means
  - Ways to increase Dependability
- Attributes
  - Way to assess Dependability
Means to achieve dependable software
Means

Dependability

- Impairments
  - Faults
  - Errors
  - Failures
   - Construction
    - Fault Prevention
    - Fault Tolerance
    - Fault Removal
    - Fault Forecasting
   - Validation
   - Security
    - Availability
    - Integrity
    - Confidentiality
- Attributes
  - Reliability
  - Safety
  - Maintainability
Means

- **Fault Prevention**
  - Prevent fault occurrence or introduction

- **Fault Removal**
  - Reduce the presence of faults

- **Fault Tolerance**
  - Ensures a service capable of fulfilling system's functions in presence of faults

- **Fault Forecasting**
  - Predicts likely faults so that they can be removed or their effects can be circumvented
Fault Prevention

- **Fault prevention techniques** are dependability enhancing techniques employed during software development to reduce the number of faults introduced during construction.

- **Fundamental techniques:**
  - Refinement of system requirements
  - Engineering software specification process
  - Structured design methods (e.g. writing clear and structured code)
  - Reusability
  - Formal Methods
System Requirements Specification

- **Imperfect process**
- **System failures may occur**
  - due to logic errors incorporated in the requirements
  - Software matches requirements, but the derived system behavior is not the expected one
  - Due to lack of communication between software and system engineering disciplines
- **Solution:** Interactive refinement of requirements and engineering of the requirements specification process
Structured Design vs Formal Methods

- **Structured Software Design** and programming reduces component's complexity and interdependency => reduces the introduction of faults
  - Decoupling and modularization
  - Information hiding
- **Formal Methods** are very thorough, using mathematically tractable languages and tools to verify correctness and appropriateness.
  - Drawback: overhead on the development process
  - Used for small components highly critical to the entire system
Reusability

- **Reusability** of code components can be helpful when the code to be reused has been proven to be dependable
  - **Drawback**: reuse of software doesn't guarantee improvement in dependability (e.g. highly reliable software is not necessarily safer)
Fault Removal

- **Fault removal techniques** involve
  - **Detecting** existing faults (through verification and validation (V&V) methods)
  - **Eliminating** existing faults

- This techniques improve system dependability through:
  - **Software Testing**
  - **Formal inspection**
  - **Formal design proofs**
Fault Removal Techniques

- **Software testing**
  - Prohibitive cost
  - Complexity of exhaustive testing over large systems
  - Testing can show the presence but not the absence of faults
  - Adequate test coverage and appropriate test quality measures
  - Efficient testing only on small and critical components

- **Formal Inspection**
  - Rigorous process, accompanied by documentation
  - Source code examination to find, correct faults and verify correction
  - Performed prior to the testing phase life cycle
Fault Removal Techniques

- **Formal design proofs**
  - Closely related to formal methods
  - Mathematical proof for correctness
  - Costly and complex to use
  - Not fully developed methods
  - Feasible on a small and critical portion of code
Fault Forecasting

- **Fault Forecasting** focuses on the reliability measure of dependability.
- Fault Forecasting techniques are used during validation to:
  - Estimate the presence of faults
  - The occurrence and consequences of failures
- This techniques include two types of activities:
  - Reliability Estimation
  - Reliability Prediction
Reliability **Estimation and Prediction**

- **Reliability Estimation**
  - Determines *current* software reliability through statistical interference techniques to failure data obtained during testing or system operation.

- **Reliability Prediction**
  - Determines *future* software reliability based upon available software metrics and measures.
  - Techniques used depend on the software development stage.
Fault Tolerance techniques enable a system to tolerate software faults remaining in the system after its development. This techniques provide service complying with the relevant specification in spite of faults through:

- Single Version Software Environment
- Multiple Version Software Environment
- Multiple Data Representation Environment
Single vs Multiple Version SE

- **Single Version SE**
  - Monitoring
  - Atomicity of actions
  - Decision Verification
  - Exception Handling

- **Multiple Version SE** (design diversity)
  - Functionally equivalent and independent software versions
  - **Examples:** Recovery Blocks (RcB), N-version programming (NVP), N-self checking programming
Redundancy for software fault tolerance

- Robust Software
  - Out of range inputs
  - Inputs of the wrong type
  - Inputs in the wrong format
Redundancy Implementation

```
Inputs

Valid Input?

True

Robust software

Invalid Input?

Request new input

or

Use last acceptable value

or

Use predefined default value

Raise exception flag

Continue software operation

Result

Handle exceptions
```
Design Diversity

- Provision of identical services through separate design and implementations
Well-known design diverse techniques are:

- Recovery Blocks (RcB)
- N-Version Programming (NVP)
- Distributed Recovery Blocks
- N Self-Checking Programming
- Consensus Recovery Block
- Acceptance Voting
Data diversity

- Three well-known data diverse techniques are:
  - Retry blocks (Amman and Knight)
  - N-Copy Programming (Ammann and Knight)
  - Two pass adjudicator (Phullum)
Reliability Measures
Software Reliability **Definition and Measure**

- Software Reliability is defined as the **probability of failure-free software operation for a specified period of time in a specified environment**
- ➞ Reliability may be used as a **measure of the system's success in providing its function properly**
Reliability Measure

- **Reliability Function** $R(t)$ is the probability that a system will be successful in the interval from time 0 to time $t$.
  - Mathematically:
    \[ R(t) = P(T > t) \text{ s.t. } t \geq 0 \]
  - $T$ is a random variable denoting the **time-to-failure**.
The time to failure random variable $T$ has a density function $f(t)$, such that:

$$f(t) = \lim_{\Delta t \to 0} P(t < T \leq t + \Delta t)$$

- $f(t)$ describes how the failure probability is spread over time
- $f(t)$ properties:
  - Non-negative
  - Total area beneath $f(t)$ is equal to one: $$\int_{0}^{\infty} f(t) \, dt = 1$$
Common pdf

- Common probability distribution functions (pdf) that have applications in reliability engineering (Pham 2000a) are:
  - Binomial Distribution
  - Poisson Distribution
  - Exponential Distribution
  - Normal Distribution
  - Weibull Distribution
- Given a particular pdf:
  - $R(t)$ can be derived directly
Availability Measure

- **Availability** $A(t)$ is defined as the probability that the system is successful at time $t$
  - Mathematically:
    
    $$A(t) = \frac{\text{System up time}}{\text{System up time} + \text{System down time}} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

- **Repairable systems**: $A(t) \geq R(t)$
- **Non-repairable systems**: $A(t) = R(t)$
**Availability: Mean Time Between Failures (MTBF)**

- **MTBF** is the expected value of the random variable time between failures defined as:

\[
MTBF = MTTF + MTTR
\]

![Diagram showing the relationship between uptime, downtime, and time between failures](image-url)
Reliability Models
Model Types

- It is highly desirable and difficult, without knowing what the initial errors are, to have an estimate of the remaining errors in a software system.

- There exist two main types of software reliability models:
  - Deterministic
  - Probabilistic
Deterministic Reliability Models
Deterministic Model

- The Deterministic Model is used to study in the program:
  - The number of distinct operators and operands
  - The number of errors and machine instructions
- Performance measures of deterministic type are obtained:
  - By analyzing the program texture
  - Do not involve any random event
Well-known models

- There exist two deterministic well-known models:
  - Halstead's software metric
  - McCabe's cyclomatic complexity metric
Halstead vs McCabe

- Halstead's software metric is used to estimate the number of errors in a program
- McCabe's cyclomatic complexity metric is used to determine an upper bound model for estimating the number of remaining software defects
- Both models represent a growing quantitative approach to the measurement of computer software
Probabilistic Reliability Models
Classification

- According to (Pham2000a) probabilistic reliability models are classified in different groups:
  - Error seeding
  - Failure rate
  - Curve fitting
  - Reliability growth
  - Markov structure
  - Time-series
  - Nonhomogeneous Poisson process
Case Study

ACS (Attitude Control System) for AraMiS satellite
Use Case Diagram of the ACS system
Class Case Diagram of the ACS system
Tools usage
Tools

- Commercial tools:
  - Lambda Predict
  - Weibull++
  - ALTA
  - DOE++
  - Etc...

- Other tools:
  - CASRE (Computer Aided Software Reliability tool)
  - AutoTest
Conclusions

- All software tolerance techniques provide tolerance to software design faults, but do not provide protection against errors in requirement specifications.
- These techniques are widely used in systems where faults can result in failures with catastrophic consequences:
  - Aerospace, Nuclear Power, Healthcare etc.
Thank you very much for the attention!
Suggestions are kindly appreciated