Department of computer Engineering
ati.ttu.ee

IAF0530/IAF9530

Süsteemide usaldusväärsus ja veakindlus
Dependability and fault tolerance

Loeng 4
Risk reduction. Testing

Gert Jervan
gert.jervan@pld.ttu.ee

Tallinn University of Technology
Department of Computer Engineering
Estonia

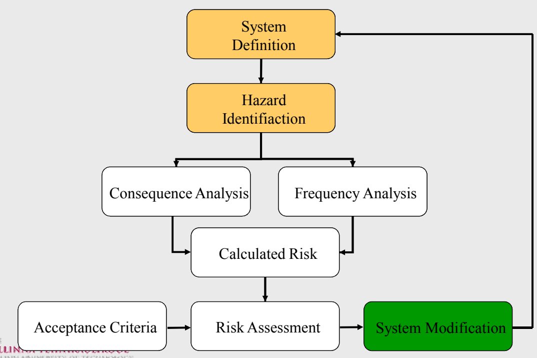
Important

- Draft of the report (incl. introductory presentation of the topic):
  - March 15
  - Drafts also by e-mail, after the meeting

Lecture Outline

- Risk Reduction & Design
- Test Economics
- Types of Testing
- Testing coverage

Hazard and Risk Analysis Process



Risk Reduction Procedures

- Four main categories of risk reduction strategies, given in the order that they should be applied:
  - Hazard Elimination
  - Hazard Reduction
  - Hazard Control
  - Damage Limitation
- Only an approximate categorisation, since many strategies belong in more than one category

Hazard Ellimination

- Before considering safety devices, attempt to eliminate hazards altogether
  - use of different materials, e.g., non-toxic
  - use of different process, e.g., endothermic reaction
  - use of simple design
  - reduction of inventory, e.g., stockpiles in Bhopal
  - segregation, e.g., no level crossings
  - eliminate human errors, e.g., for assembly of system use colour coded connections

Design Principles

- Familiar
  - use tried and trusted technologies, materials techniques
- Simple
  - testable (including controllable and observable)
  - portable (no use of sole manufacturer components compiler dependent features)
  - understandable (behaviour can easily be from implementation)
  - deterministic (use of resources is not random)
  - predictable (use of resources can be predicted)
  - minimal (extra features not provided)

Design Principles (cont.)

- Structured design techniques
  - defined notation for describing behaviour
  - identification of system boundary and environment
  - problem decomposition
  - ease of review
- Design standards
  - limit complexity
  - increase modularity
- Implementation standards
  - presentation and naming conventions
  - semantic and syntactic restrictions in software

Classes of System Failure

- Random (physical) failures
  - due to physical faults
  - e.g., wear-out, aging, corrosion
  - can be assigned quantitative failure probabilities
- Systematic (design) failures
  - due to faults in design and/or requirements
  - inevitably due to human error
  - usually measured by integrity levels
- Operator failures
  - due to human error
  - mix of random and systematic failures

Nature of Random Failures

- Arise from random events generated during operation or manufacture
- Governed by the laws of physics and cannot be eliminated
- Modes of failure are limited and can be anticipated
- Failures occur independently in different components
- Failure rates are often predictable by statistical methods
- Sometimes exhibit graceful degradation
- Treatment is well understood

Treating Random Failures

- Random failures cannot be eliminated and must be reduced or controlled
- Random failures can be mitigated by:
  - predicting failure modes and rates of components
  - applying redundancy to achieve overall reliability
  - performing preventative maintenance to replace components before faults arise
  - executing on-line or off-line diagnostic checks

Nature of Systematic Failures

- Ultimately caused by human error during development, installation or maintenance
- Appear transient and random since they are triggered under unusual, random circumstances
- Systematic and will occur again if the required circumstances arise
- Failures of different components are *not* independent
- Difficult to predict mode of failure since the possible deviations in behaviour are large
- Difficult to predict the likelihood of occurrence

Treating Systematic Failures

- In theory, design failures can be eliminated
- In practice, perfect design may be too costly
- Focus the effort on critical areas
  - identify safety requirements using hazard analysis
  - assess risk in system and operational context
- Eliminate or reduce errors using quality development processes
  - verify compliance with safety requirements
  - integrate and test against safety requirements

Design Faults

- Design faults are much more difficult to deal with than random (degradation) faults because:
  - They are hard to anticipate
  - Their effects are hard to predict
  - Component failure semantics tend to be undefined
- This makes all forms difficult to tolerate, especially software faults

Common Design Faults

- All forms of software:
  - System software
  - Application software
  - Embedded software (firmware)
- All forms of computing hardware:
  - Hardware design faults now dominate
  - Degradation faults used to dominate
- Power supply systems
- Component interconnection wiring

Design Diversity

- Idea:
  - Design faults are “aspects” of design
  - Different designs, different faults
  - Produce multiple designs—independent level.
  - Operate in parallel at execution time
- Applies to all types of design fault
- Can be configured using many system architectures, like NMR, TMR, etc.

Hazard Reduction

- Reduce the likelihood of hazards
- Use of barriers, physical or logical
  - Lock-ins
  - Lock-outs
  - Interlocks
- Failure minimization
  - Redundancy
  - Recovery

Forms of Redundancy

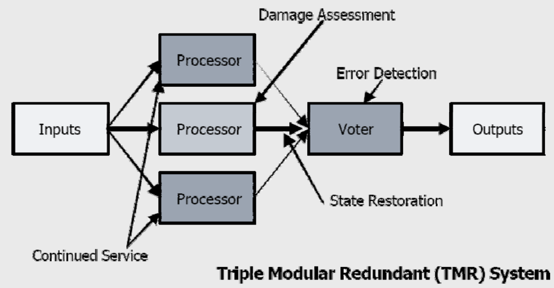
- Hardware redundancy
- Software redundancy
- Information redundancy
- Temporal (time) redundancy

- Design diversity, for hardware/software
  - Develop different implementations of the same hardware/software component
  - Called N-version programming
  - Then apply static or dynamic redundancy

Hardware Redundancy

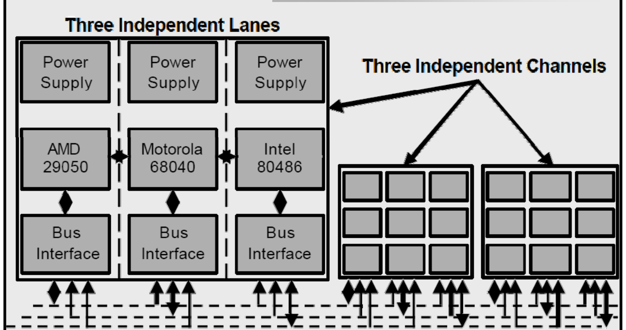
- Static redundancy
  - Component (at least) triplicated
    - Triple Modular Redundancy (TMR), N-Modular Redundancy (NMR)
  - Voting element used to remove effects of single failure
  - Loss Of Unit Implies:
    - Removal Or Containment
    - Service Provided By Those That Remain
- Dynamic redundancy
  - Component has a mirror that is invoked when fault occurs
  - Cold or Hot Standby, spares
  - Loss Of Unit Implies:
    - Removal Or Containment
    - Introducing Standby Unit

Fault Tolerant System Example



Risk of single-point failure

B777 Primary Flight Computer Architecture



N Modular Redundancy

- Independent development of modules
- This is what Boeing did with N = 3 for processors
- Operation:
  - Parallel—forward error recovery
  - Serial—backward error recovery
- In software with forward error recovery, referred to as N-version programming
- In software with backward error recovery, referred to as recovery block

B777 PFC CPUs

- Problem:
  - Processors often (essentially always) contain design faults, need to deal with them
  - 777 channel is a TMR system
- Three manufacturers, three designs
- Are these designs different?
- How would you measure the difference?
- What metric is there for design diversity?

Redundancy

- Software redundancy, e.g. N-version programming
- Information redundancy, e.g., checksums, cyclic redundancy codes, error correcting codes
- Hybrid redundancy

N-Version Programming

- NMR for software
- Practical issues:
  - Cost of development, team separation
  - Resources during execution
  - Different execution times for different versions
  - Different but similar output values
  - Different but valid output values (multiple correct solutions)

N-Version Programming

- Performance:
  - Assumed statistical independence
  - If not independent, then no lower bound
  - Common specification defects
  - Common implementation (design) faults
- Problem compounded by comparison checking during testing

Hybrid Redundancy

- N-S modular redundancy with "S" spares
- As members of the N-S fail, spares switched in
- Able to tolerate up to N-2 failures
- Spares may be unpowered:
  - Saves power
  - Unpowered units much more reliable than powered
  - Attention required to infant mortality
- Clearly applicable to:
  - Long-duration systems
  - Systems with no repair opportunity

Space Shuttle Computer System

- Uses combination of
  - Redundancy, fault detection and design diversity
- Hardware voting on sensors and actuators
- Five identical computers
  - During critical stages, four computers work in NMR with voting for fault detection
  - Fifth computer performs non-critical functions, e.g. comm.
- Fault tolerance
  - Tolerates failure of two computers
  - In case of third failure, crew/ground control decide which computer wins
  - Fifth computer can take over control, uses different software

Recovery

- Can reduce failures by recovering after error detected but before component or system failure occurs
- Recovery can only take place after detection of error
  - Backward recovery
  - Forward recovery

Error Detection

- Based on check that is independent of implementation of the system
  - coding - parity checks and checksums
  - reasonableness - range and invariants
  - reversal - calculate square of square root
  - diagnostic - hardware built-in tests
  - timing - timeouts or watchdogs

Error Detection (cont.)

- Timing of error detection important
  - early error detection can be used to prevent propagation
  - late error detection requires a check of the entire activity of system
- Checking may be in several forms
  - monitor, acting after a system function, checking outputs after production but before use
  - kernel, encapsulating (safety-critical) functions in a subsystem that allows all inputs to and outputs from the kernel to be checked

Backward Recovery

- Corrects errors through reversing previous operations
- Return system to a previous known safe state
- Allows retry
- Requires checkpoints or saved states (and the expenses involved with producing them)
- Rollback usually impossible with real-time system

Forward Recovery

- Corrects errors without reversing previous operations, finding safe (but possibly degraded) state for system
  - data repair, use redundancy in data to perform repairs
  - reconfiguration, use redundancy such as backup or alternate systems
  - coasting, continue operations ignoring (hopefully transient) errors
  - exception processing, only continue with selection of (safetycritical) functions
  - failsafe, achieve safe state and cease processing
    - use passive devices (e.g., deadman switch) instead of active devices (e.g., motor holding weight up)

Hazard Control

- Detect and control hazard before damage occurs
- Reduce the level or duration of the hazard
- Hazard control mechanisms include:
  - Limiting exposure: reduce the amount of time that a system is in an unsafe state (e.g. don't leave rocket in armed state)
  - Isolation and containment
  - Fail safe design

Damage Limitation

- In addition to eliminating hazards or employing safety devices, consider
  - warning devices
  - procedures
  - training
  - emergency planning
  - maintenance scheduling
  - protective measures

Architectural Design

- Suitable architectures may allow a high integrity system to be built from lower integrity components
  - combinations of components must implement a safety function independently
  - overall likelihood of failure should be the same or less
  - be wary of common failure causes
- Apportionment approaches can be quantitative and/or qualitative
  - quantitative: numerical calculations
  - qualitative: judgement or rules of thumb

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Department of computer Engineering
ati.ttu.ee

Fault Tolerance

Basics

- Computing systems are characterized by five fundamental properties:
  - functionality
  - usability
  - performance
  - cost
  - dependability

Faults

- Faults are there!
- Either prevent, **tolerate**, remove or forecast
- We need redundancy
  - System that is more complex than needed for performing the required task

Means to Achieve Dependability

- Fault prevention
  - Good design processes, avoid design flaws
  - Good procedures for runtime faults
- Fault tolerance
  - Fault detection
  - Redundancy
  - Diversity
- Fault removal
  - Verification and validation during design
  - Corrective/preventive action during maintenance
- Fault forecasting
  - Simulation, modelling, prediction
  - Analysis based on history statistics

Fault Tolerance

- Automobile:
  - Spare Tires
  - Dual Braking Systems
- Power Supplies:
  - UPS/battery backup
  - Power-fail interrupts
- Multiple engines on aircraft
- Emergency lighting in buildings
- Tape backups of disk files
- Checkpoint/restart of long-running programs
- Parity and SECDED in computer memories

Faults

- Random faults (Degradation faults)
  - Arise during operation
  - Usually hardware component failure
- Systematic faults (Design Faults)
  - mistakes in the spec
  - mistakes in the hardware
  - mistakes in the software

Faults

- Faults are either permanent, transient or intermittent
- Design faults are always permanent
- Dealing with faults:
  - During development: fault avoidance & removal
  - During operation: fault tolerance & detection

Hardware Faults

- Use of fault models
- Decomposition into modules
  - Gates, transistors, etc
- Connection faults
  - Single stuck-at model, bridging model (shorts), stuck-open
- Used to model hardware faults
  - Design testing schemes for digital circuits
  - Fault removal coverage usually less than 100%
  - Guard against physical defects, not design faults
- In safety critical systems
  - Combined with Failure Modes and Effects Analysis (FMEA)
  - Need fault avoidance by verification…

Other Faults

- Hardware design and specification faults
  - Few fault models available
  - Many faults cannot be modelled
  - System must meet the spec, but spec might be incorrect as well
  - Spec errors may manifest as either hardware or software failures
  - Use of formal methods (formal spec. languages, automata theory, formal verification, model checking, etc.)

Software Faults

- Bugs:
  - Software spec faults
  - Coding faults
  - Logical errors within calculations
  - Stack overflows or underflows
  - Uninitialized variables
- No random failures and it does not degrade with age
- Always systematic
- Exhaustive testing almost impossible
- Must be tolerated

SW Testing - i.e. Verification

- Verification:
  - SW testing
  - formal verification
- Functional and structural testing
- Path testing, transaction flow testing, data-flow testing, domain testing, mutation testing etc.

Fault Detection Techniques

- Functionality checking
  - march test
- Consistency checking
  - range checking, overflow
- Signal comparison
- Information redundancy
  - checksums, cyclic redundancy codes, error correcting codes
- Monitoring techniques
  - Loopback testing
  - Power supply monitoring

Watchdog Timer

- An inexpensive method of error detection
- Process being watched must reset the timer before the timer expires, otherwise the watched process is assumed as faulty
- Watchdog timers only detect errors which manifest themselves as a control-flow error such that the system does not continue to reset the timer
- Only processes with relatively deterministic runtimes can be checked, since the error detection is based entirely on the time between timer resets

Heartbeats

- A common approach to detecting process and node failures in a distributed (networked) computing environment.
- Periodically, a monitoring entity sends a message (a heartbeat) to a monitored node or process and waits for a reply.
- If the monitored node does not respond within a predefined timeout interval, the node is declared as failed and appropriate recovery action is initiated.
- Adaptive or smart

System Testing

HW Testing SW Testing

HW/SW Testing
(system testing)

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY
Department of computer Engineering
ati.ttu.ee

Software Testing

1918 TALLINNA TEHNIKAÜLIKOOL TALLINN UNIVERSITY OF TECHNOLOGY
Department of computer Engineering
ati.ttu.ee

Programmers are in a race with the Universe to create bigger and better idiot-proof programs.

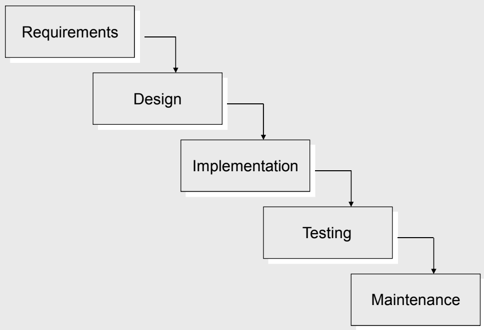
While the Universe is trying to create bigger and better idiots.

So far the Universe is winning

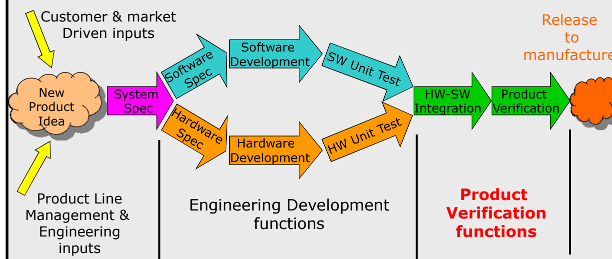
Software Testing Topics

- Test Economics
- Types of Testing
- Testing coverage

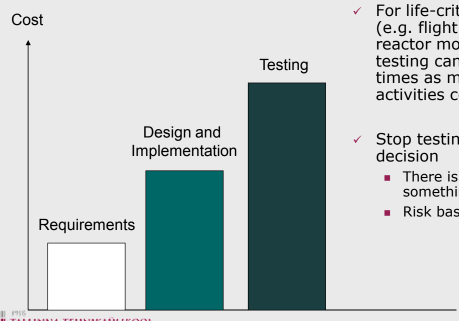
Software Life Cycle



The Product Development Cycle

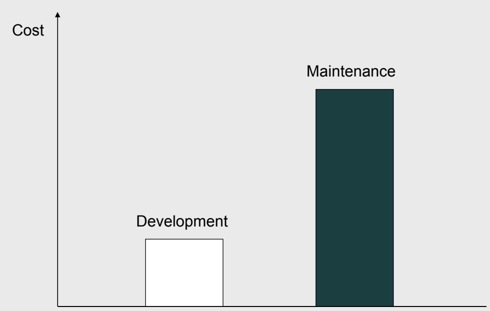


Software Development Costs



- For life-critical software (e.g. flight control, reactor monitoring), testing can cost 3 to 5 times as much as all other activities combined.
- Stop testing is a business decision
  - There is always something more to test
  - Risk based decision

Software Life Cycle Costs



Software Qualities

- Correctness
- Reliability (dependability)
- Robustness
- Safety
- Security (survivability)
- Performance
- Productivity
- Maintainability, portability, interoperability, …

Software Verification and Validation

- Verification
  - Are we building the product right?
  - Process-oriented
    - Does the product of a given phase fulfill the requirements established during the previous phase?
- Validation
  - Are we building the right product?
  - Product-oriented
    - Does the product of a given phase fulfill the user's requirements?

Techniques for V&V

- ✓ Static
  - Collects information about a software without executing it
    - Reviews, walkthroughs, and inspections
    - Static analysis
    - Formal verification
- ✓ Dynamic
  - Collects information about a software with executing it
    - Testing: finding errors
    - Debugging: removing errors

Static Analysis

- ✓ Control flow analysis and data flow analysis
  - Extensively used for compiler optimization and software engineering
- ✓ Examples
  - Unreachable statements
  - Variables used before initialization
  - Variables declared but never used
  - Variables assigned twice but never used between assignments
  - Variables used twice with no intervening assignment
  - Possible array bound violations

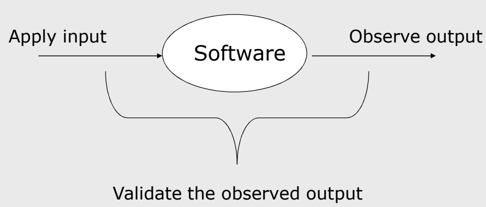
Formal Verification

- ✓ Given a model of a program and a property, determine whether the model satisfies the property based on mathematics
- ✓ Examples
  - Safety
    - If the light for east-west is green, then the light for south-north should be red
  - Liveness
    - If a request occurs, there should be a response eventually in the future

Introduction to Testing

- ✓ Debugging and testing are not the same thing!
- ✓ Testing is a systematic attempt to break a program.
  - Correct, bug-free programs by construction are the goal but until that is possible (if ever!) we have testing.
  - Since testing is basically **destructive** in nature, it requires that the tester discard *preconceived* notions of the *correctness* of the software to be tested

Testing



Is the observed output the same as the expected output?

Software Testing Fundamentals

- ✓ Testing objectives include
  - Testing is a process of executing a program with the intent of finding an error.
  - A **good** test case is one that has a high probability of finding an as yet undiscovered error.
  - A **successful** test is one that uncovers an as yet undiscovered error.

Limitations of Testing (I)

- To test all possible inputs is impractical or impossible

```
int foo(int x) {
    y = very-complex-computation(x);
    write(y);
}
```

- To test all possible paths is impractical or impossible

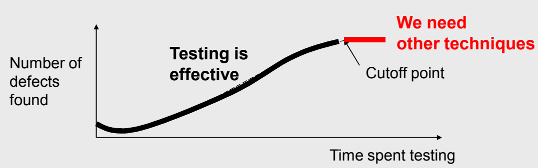
```
int foo(int x) {
    for (index = 1; index < 10000; index++)
        write(x);
}
```


Limitations of Testing (II)

- Dijkstra, 1972
  - Testing can be used to show the presence of bugs, but never their absence
- Goodenough and Gerhart, 1975
  - Testing is successful if the program fails
- The (modest) goal of testing
  - Testing cannot guarantee the correctness of software but can be effectively used to find errors (of certain types)

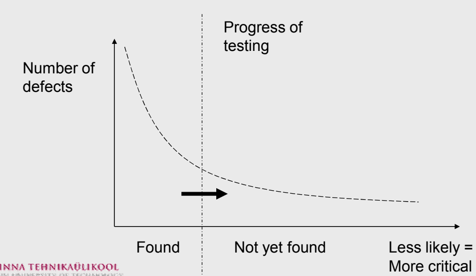
Economics of Testing (I)

- The characteristic S-curve for error removal



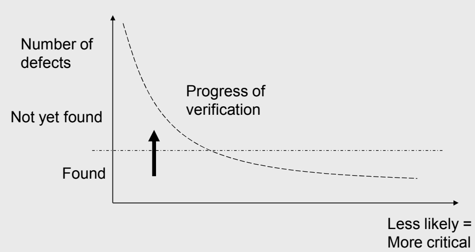
Economics of Testing (II)

- Testing tends to intercept errors in order of their probability of occurrence



Economics of Testing (III)

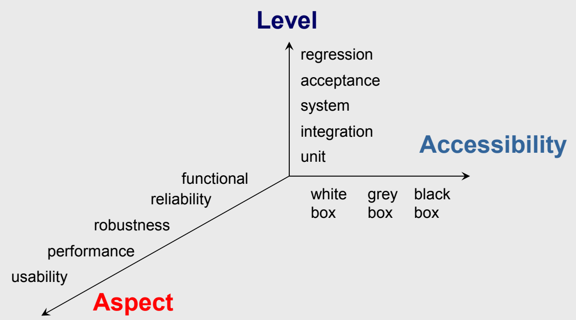
- Verification is insensitive to the probability of occurrence of errors



Fundamental Questions in Testing

- When can we stop testing?
  - Test coverage
- What should we test?
  - Test generation
- Is the observed output correct?
  - Test oracle
- How well did we do?
  - Test efficiency
- Who should test your program?
  - Independent V&V

Types of Testing



Important

- ✓ Draft of the report (incl. introductory presentation of the topic):
  - March 15
  - Drafts also by e-mail, after the meeting

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
Department of computer Engineering
ati.ttu.ee

Questions?