IAF0530/IAF9530

Dependability and fault tolerance

Lecture 4
Fault Tolerance, Software Testing

Gert Jervan
gert.jervan@pld.ttu.ee



## Important

- No lecture on March 13!
- March 20
  - Draft of the report (by e-mail, before the lecture)
    - Abstract, outline, main references, ca. 1 page
  - Elevator pitch (max 5 min, 2-3 slides).
  - Slides to be sent by e-mail at least 1 hour before the lecture!
  - 26 presentations – sharp timing is mandatory!
  - Participation mandatory!




# 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


7

## 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


8

## 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


9

## 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…


10

## 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.)


11

## 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


12

## 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.


13

## 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


14

## 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


15

## 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


16

## System Testing

HW Testing SW Testing

HW/SW Testing
(system testing)


17

## Software Testing

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



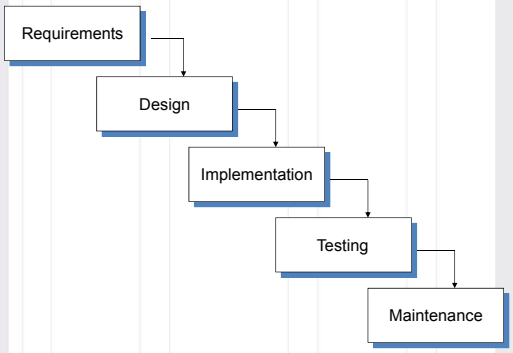
## Software Testing Topics

- Test Economics
- Types of Testing
- Testing coverage





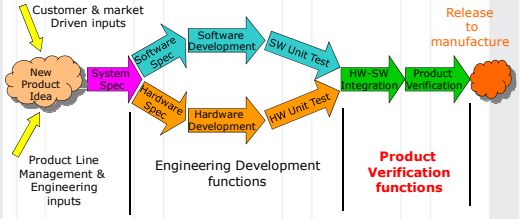
## Software Life Cycle







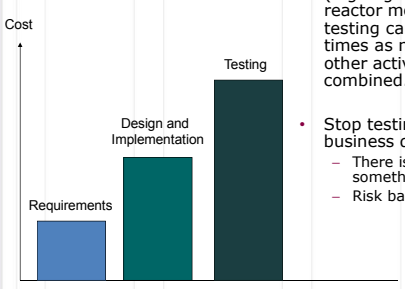

## The Product Development Cycle








## Software Development Costs

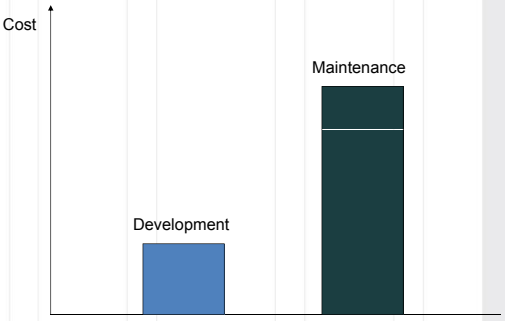



- For life-critical software (e.g. flight control, reactor monitoring), testing can cost 3 to 5 times as much as all other activities combined.
- Stop testing is a business decision
  - There is always something more to test
  - Risk based decision





## Software Life Cycle Costs



## Software Qualities

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


25

## 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?


26

## 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


27

## 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


28

## 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

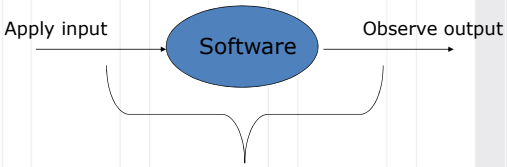

29

## 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


30

## Testing




Validate the observed output

Is the observed output the same as the expected output?


31

## 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.


32

## 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);
}
```


33

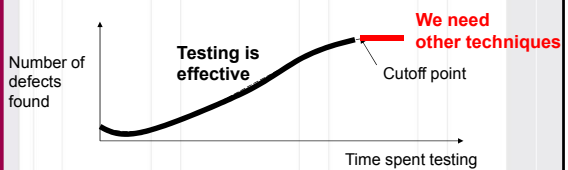
## Limitations of Testing (II)

- 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)


34

## Economics of Testing (I)

- The characteristic S-curve for error removal

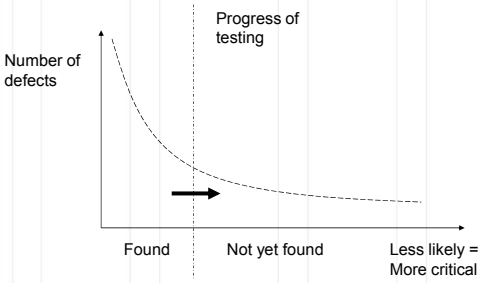




35

## Economics of Testing (II)

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

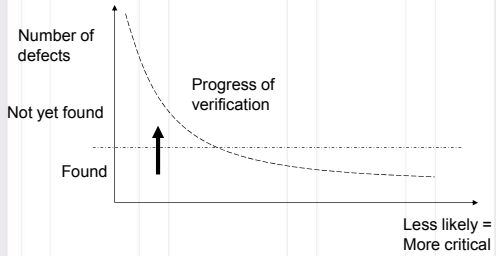




36

## Economics of Testing (III)

- Verification is insensitive to the probability of occurrence of errors





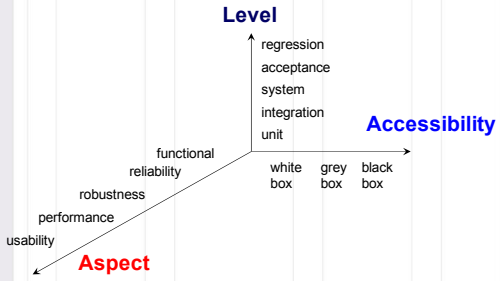
37

## 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


38

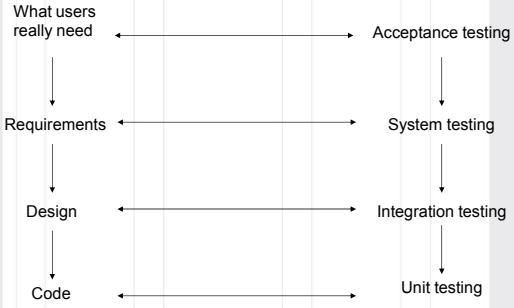
## Types of Testing





39

## Levels of Testing





40

## Accessibility of Testing

- White box testing (structural testing, program-based testing)
- White box testing is a test case design method that uses the control structure of the procedural design to derive test cases. Test cases can be derived that
  - guarantee that all independent paths within a module have been exercised at least once,
  - exercise all logical decisions on their true and false sides,
  - execute all loops at their boundaries and within their operational bounds, and
  - exercise internal data structures to ensure their validity.


41

## Accessibility of Testing (II)

- Black box testing (functional testing, specification-based testing)
  - Assumes that the program is unavailable or testers do not want to look at the details of the program
    - Derives test cases from the requirements of the program
    - Controls and observes the program only through external interfaces
    - Ideally done by independent test group (not original programmer)
- Grey box testing

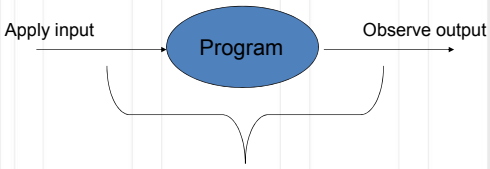

42

## Program-Based Testing (I)

- Main steps
  - Examine the internal structure of a program
  - Design a set of inputs satisfying a coverage criterion
  - Apply the inputs to the program and collect the actual outputs
  - Compare the actual outputs with the expected outputs
- Limitations
  - Cannot catch omission errors
    - What requirements are missing in the program?
  - Cannot provide test oracles
    - What is the expected output for an input?




## Program-Based Testing (II)




Validate the observed output against the expected output

Who will take care of test oracles?




## Covergae metrics

- Statement coverage
- Branch coverage
- Path coverage
- Mutation coverage

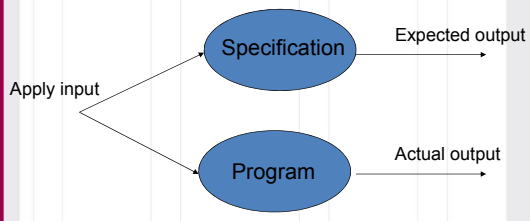



## Specification-Based Testing (I)

- Main steps
  - Examine the structure of the program's specification
  - Design a set of inputs from the specification satisfying a coverage criterion
  - Apply the inputs to the specification and collect the expected outputs
  - Apply the inputs to the program and collect the actual outputs
  - Compare the actual outputs with the expected outputs
- Limitations
  - Specifications are not usually available
    - Many companies still have only code, there is no other document.




## Specification-Based Testing (II)




Validate the observed output against the expected output




## The Budget Coverage Criterion

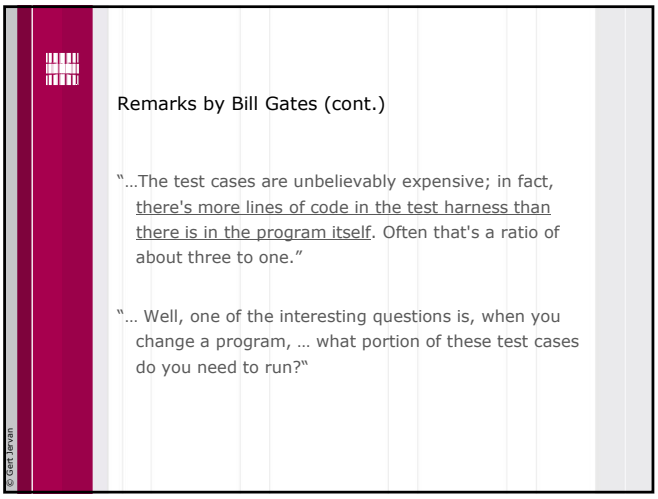
- A common answer to "when is testing done"
  - When the money is used up
  - When the deadline is reached
- This is sometimes a rational approach!
  - Implication 1: Test selection is more important than stopping criteria per se.
  - Implication 2: Practical comparison of approaches must consider the cost of test case selection

Remarks by Bill Gates
17th Annual ACM Conference on Object-Oriented Programming, Seattle, Washington, November 8, 2002

"… When you look at a big commercial software company like Microsoft, there's actually as much testing that goes in as development. We have as many testers as we have developers. Testers basically test all the time, and developers basically are involved in the testing process about half the time…

… We've probably changed the industry we're in. We're not in the software industry; we're in the testing industry, and writing the software is the thing that keeps us busy doing all that testing."





Remarks by Bill Gates (cont.)

"…The test cases are unbelievably expensive; in fact, there's more lines of code in the test harness than there is in the program itself. Often that's a ratio of about three to one."

"… Well, one of the interesting questions is, when you change a program, … what portion of these test cases do you need to run?"