DD2459

Software Reliability

Karl Meinke

karlm@nada.kth.se

Lecture 1: Introduction.

Course Material

- This course is mainly about software testing
- We cover the whole testing activity
- We emphasize test design as a practical skill
- We consider classical and modern approaches to testing
- We don’t have time to cover everything!
- We might consider reliability models.

Course Format

- 7 lectures – fundamental theory + math
- 7 exercise classes: alternate weeks
  - 1 week practical instructions
  - 1 week practical lab work
- Labs are mini projects (3 points)
  - Work in pairs
  - Build or use existing tools
- Short take-home exam (4.5 points)

Course Material

Amman and Offut, *Introduction to Software Testing*, Cambridge University Press, 2nd edition, 2016.

Jorgensen, *Software Testing: a Craftsman's Approach*, Auerbach Publications, 2008.

What is Testing?

Black-box testing	Load Testing
Regression testing	Security testing
Functional testing	Pairwise testing
Random testing	Unit testing
Alpha/Beta testing	Integration testing
Acceptance testing	System testing
Performance testing	Usability testing

Some Views …

- *“Testing can show the presence of bugs but not their absence”* (Dijkstra).

- *“Testing is an infinite process of comparing the*
  bugs/error what the code should do
  invisible to the ambiguous in order to avoid
  software disastrous e.g Boeing 747 software practices customers
  the unthinkable happening to the anonymous”

- (*James Bach [3]*)

… And Definitions

before during

after

- Testing concerns the design, execution and subsequent analysis of individual test cases to evaluate a system.
- Testing concerns dynamic code execution (in situ) rather than static analysis

  can be much powerful than testing,
  but more restrictive in domain

  e.g memory leakage in C++, race condition, division by zero
- Testing has different goals according to one's level of test maturity.

  e.g economic, business goals

IEEE SWEBOK 2004

- Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems …

in static analysis, we don't execute the code --> cannot see certain things

- Software testing consists of the *dynamic* verification of the behavior of a program on a *finite* set of test cases, suitably *selected* from the usually infinite executions domain, against the *expected* behavior.

can only execute a small finite set of test cases

therefore, need to be suitably selected

- www.swebok.org

How to study?

- Bewildering variety of themes
- Try to find similarities of approach
- Reusable concepts and techniques
  - E.g. graph models and coverage models
- Focus on functional (behavioural) testing
- Focus on test design

With limited test cases.
Optimisation of test cases design is required.

- Use lab work to focus our studies

The Price of Failure

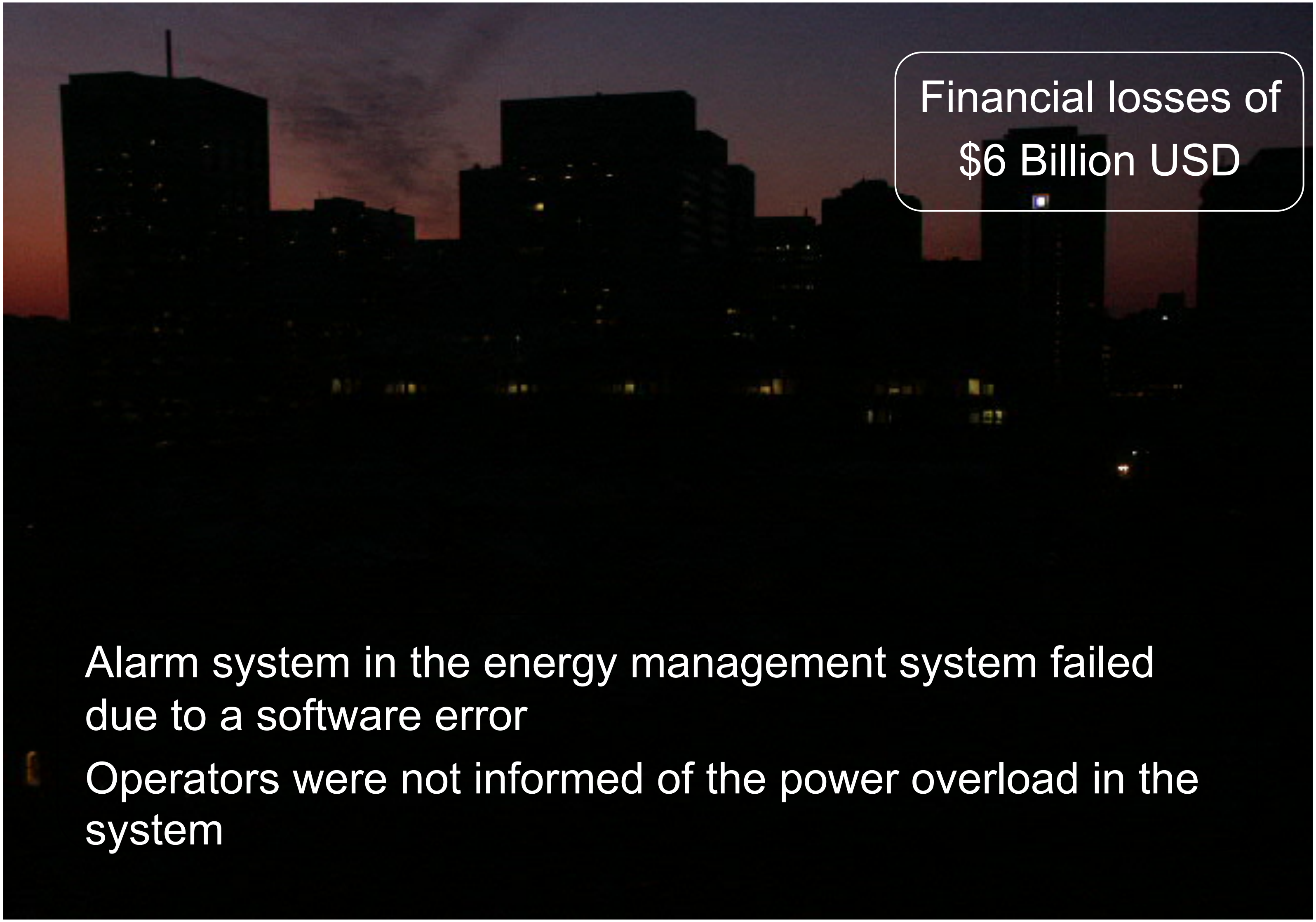
- NIST report (2002) – inadequate testing costs USA alone $22 - $59 billion dollars annually
- Improved testing could half this cost
- Web application failures
  - Financial services $6.5 million per hour
  - Credit card sales applications $2.4 million per hour
- Symantec: most security vulnerabilities due to software errors.

- NASA Mars lander (1999) due to an integration error
- Ariane 5 explosion – exception handling bug, due to outdated requirement. $370 million
- Toyota brakes: dozens dead, thousands of crashes
- Therac-25 radiation machine: three dead
- Major failures: Mars polar lander, Intel's pentium bug

- **Airbus 319 – fly-by-wire concept**

Loss of autopilot, flight deck lighting, primary flight and navigation displays!

- **USA Northeast Blackout 2003**



Financial losses of $6 Billion USD

Alarm system in the energy management system failed due to a software error

Operators were not informed of the power overload in the system

Conclusions

- *Software is a skin that surrounds our civilisation* (Amman and Offut)
- We need software to be reliable
- Testing is main method to assess reliability
- Testing is becoming more important
- Resources (manpower) for testing increases linearly
- Complexity of software increases exponentially
- Automated testing is inevitable

(Traditional) Test Activities – 4 Types

- Test design
  - Criteria based classic computer science approach --> goal can be measured /define/ mechanical
    --> focus of the course
  - Human based by thinking about human's behaviour --> good guess of where bugs are at
    non-teachable
- Test automation loop based in batch
- Test execution
- Test evaluation
- Need different skills, background knowledge, education and training.

1.a. Test Design – Criteria based

- Design test values to satisfy coverage criteria or other engineering goal
- Testing is a search problem, coverage measures search effort
- Most technical and demanding job of all
- Needs skills in
  - Discrete math
  - Programming
  - Testing
- Traditional Computer Science Degree

1.b. Test Design – Human based

- Design test values based on domain knowledge of program and human knowledge of testing
- Criteria based approaches can be blind to situations
- Requires knowledge of domain, testing and user interfaces
- No traditional CS required

Human-based (cont)

- Background in the software domain is essential
- Empirical background is helpful (biology, psychology etc)
- A logic background is helpful (law, philosophy, math)
- Work is experimental and intellectually stimulating.

2. Test Automation

- Embed test values into executable scripts
- Straightforward programming
  - Small pieces, simple algorithms
  - Junit, JBehaviour
- Needs little theory
- Little boring
- Who determines and embeds the expected outputs?
- What if system is non-deterministic?

3. Test Execution

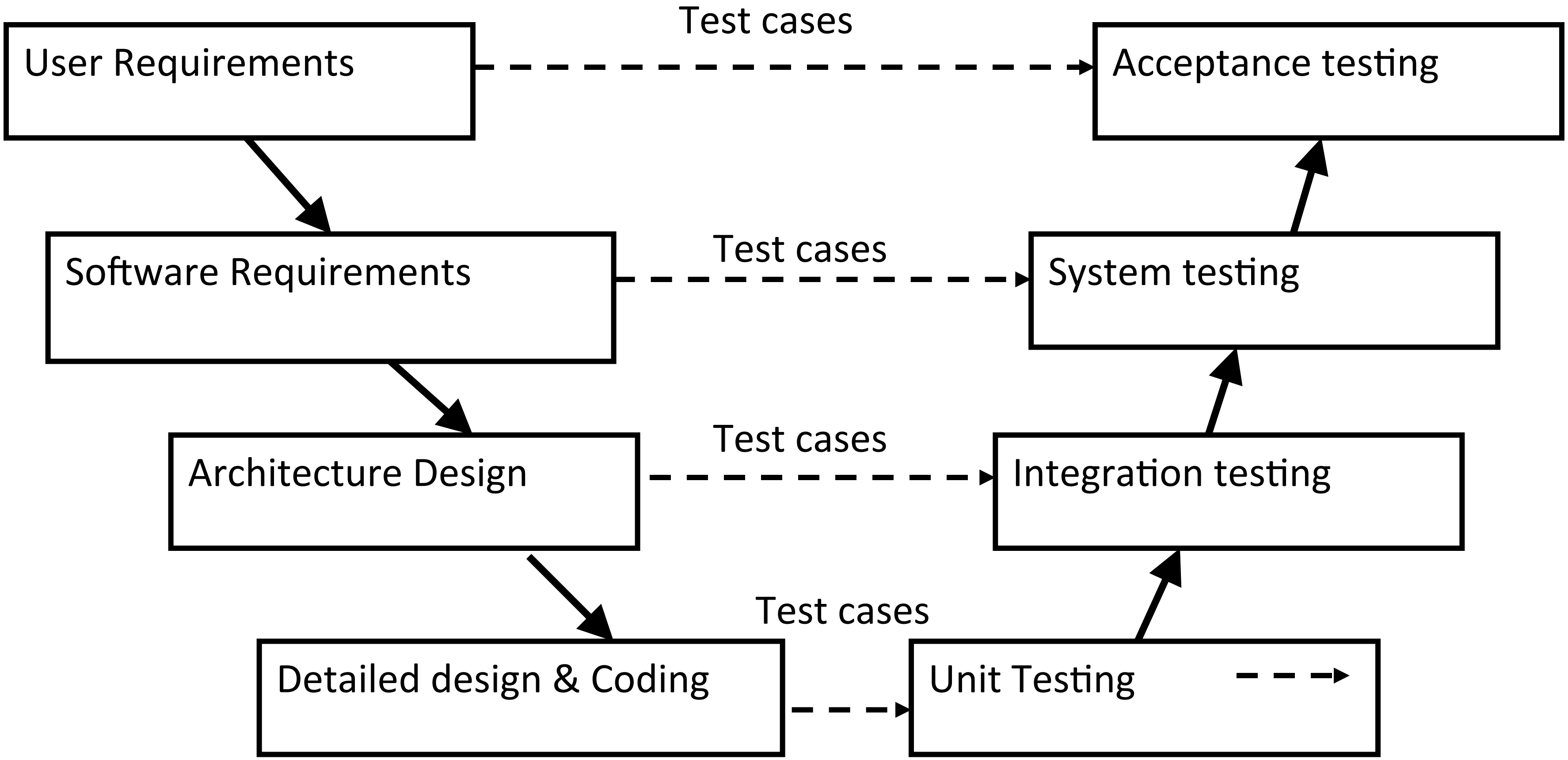
- Run tests on the SUT and record results
- Easy and trivial if tests automated
- Very junior personnel
- Test executors must be careful and meticulous with book-keeping (e.g. time of day error?)
- A test is an experiment in the real world.

4. Test Evaluation

- Evaluate outcome of testing and report to developers
- Test report
- Psychological problems – blame etc
- Test goals must be clear to assist debugging

Other Activities

- Test management: policy, group structure, integration with development, budget, scheduling.
- Test maintenance: test reuse, repositories, continuous integration, historical data, statistics, regression testing.
- Test documentation:
  - Document “why” criteria
  - Ensure traceability to requirements or architectural models.
  - Evolve with the product.



User Requirements

Test cases

Acceptance testing

Software Requirements

Test cases

System testing

Architecture Design

Test cases

Integration testing

Test cases

Detailed design & Coding

Unit Testing

The ”V” model of Testing

Integrates testing with waterfall lifecycle

Time

Test Maturity Model (TMM)

- Level 0: no difference between testing and debugging
- Level 1: purpose is to show software works
- Level 2: purpose is to show software fails
- Level 3: purpose is to reduce risk of use
- Level 4: purpose is a mental discipline for quality.

Formal Definitions

what we need to fix

1. Software Fault: *a static defect in software*
2. Software Error: *an incorrect internal state manifesting a fault*

what has happened

perform test to see

3. Software Failure: *External incorrect behaviour wrt requirements.*

what we see

Patient has a symptom of thirst (3), doctor finds high blood glucose (2), doctor diagnoses diabetes (1) part of fault analysis

- Test Failure: *execution resulting in failure*
- Debugging: *process of locating fault from failure* translation of 3 to 1
- Test case values: *input values needed to complete execution of the SUT* need at least a certain number of values to do something
- Expected results: *results that should be produced iff SUT meets its requirements*

some expectation is required to say if a test passes/fails

- Prefix (setup) values: *input necessary to bring SUT to an appropriate state to receive test case values*
- Postfix (teardown) values: *input needed to be sent after the test case values*

  reset certain values before another test case in the test suite is conducted. Put the software to default neutral state

  – Verification values: *needed to recover the results*

  – Exit values: *needed to terminate or return to a stable state*.

Def^n: Test Case

- Test Case: *the test case values, setup values, teardown values and expected values needed for one observation of the SUT.* system under test

  ***Note**: a test case should ideally bring a program to termination if possible.*

- Test Suite: *a set of test cases.*
- Dead Code: code which can never be executed by any test case (aka. unreachable code).

e.g while false
if false

Coverage

- Test requirement: *A specific (structural) element r of an SUT that a test case must cover.*
- Eg: *lines, paths, branches, variable values*.
- Coverage Criterion: *a set of rules that impose test requirements on a test suite*, e.g. node coverage
- Coverage: *Given a set R of test requirements coming from a criterion C, a test suite T satisfies C iff for each $r \in R$ there exists at least one*

  $t \in T$ which satisfies r.

Varieties of System Under Test

- Procedural (C code, FORTRAN, etc)
  - Precondition and postcondition
- Reactive (ATM machine, fly-by-wire)
  - “always on” - event driven behaviour
- Real-time (soft/hard)
- Communications protocol
- Numerical (approximately correct)
- Object-oriented (class and method invariants)
- Distributed system (non-deterministic)
- GUI, user event generation must be simulated