Software verification and validation. Introduction

Marius Minea

October 1, 2015

Topics be discussed

Black-box testing (no source access)
Glass-box/white-box testing (with source access)
- Generating unit tests
- Test coverage metrics

Static analyis (of source code)
Dynamic analysis
Testing object-oriented programs
Testing concurrent programs
Formal verification of programs/models
Automated test generation
- including Model-based testing

Security testing
Design of a test plan

High-profile errors: Therac-25

- medical system for radiation thrapy
- 6 accidents with fatalities and grave wounds (1985-87, US, Canada)
- direct cause: errors in control program

High-profile errors: Therac-25

- medical system for radiation thrapy
- 6 accidents with fatalities and grave wounds (1985-87, US, Canada)
- direct cause: errors in control program

Retrospective analysis [Leveson 1995]:
excessive trust in software when designing system
reliability $\neq$ safety
lack of hardware safety measures (interlocks)
lack of appropriate software engineering practices (defensive design, specification, documentation, simplicity, formal analysis, testing)
correcting one error does not necessarily make system safer !

Errors: Ariane 5 space rocket

- Self-destruction after a fault 40 seconds from launch (1996)
- Cause: conversion of 64-bit float to 16-bit int generated unhandled overflow exception in the ADA program
- Cost: $500 million (rocket), $7 billion (whole project)

Errors: Ariane 5 space rocket

- Self-destruction after a fault 40 seconds from launch (1996)
- Cause: conversion of 64-bit float to 16-bit int generated unhandled overflow exception in the ADA program
- Cost: \$500 million (rocket), \$7 billion (whole project)

Retrospective analysis
main cause: inappropriate software reuse
code taken from the Ariane 4, without proper analysis:
- execution of faulty code no longer needed at that flight stage
- Ariane 4 had proved absence of overflow for unprotected variables, but new settings were different

⇒ need to specify and conform to an interface
bad design of redundance: the inertial reference system and its backup were taken out by the same error

The Pentium bug

Error in the floating-point division algorithm
SRT division algorithm, base 4
determines the next quotient digit from a lookup table
some entries erroneously marked as “don’t care”
cost: some $500 million

The Pentium bug

Error in the floating-point division algorithm
SRT division algorithm, base 4
determines the next quotient digit from a lookup table
some entries erroneously marked as “don’t care”
cost: some $500 million

Retrospective analysis

Circuit could have been verified formally
- by automated theorem proving, or
- with special data structures to represent multiplication and division

but verification effort was focused on more complex parts
(execution unit, cache coherence protocol)

The Mars space probes

Mars Pathfinder, 1997
once on Mars, the spacecraft was frequently resetting
cause: priority inversion between processes with common resources
the problem and solution had been described in the literature
[Sha, Rajkumar, Lehoczky. Priority Inheritance Protocols, 1990]

1. low priority process A acquires resource R
2. A interrupted by C (high priority)
3. C awaits availability of R; A resumes execution
4. A interrupted by long-running B (priorities: $A < B < C$)

$\Rightarrow$ high-priority C waits for B, without depending on B !

Solution: a process (A) that acquires a resource has its priority raised to the highest of any process (C) that might request the resource

The Mars space probes

Mars Climate Orbiter, 1998
disintegrated on entry to Mars atmosphere
error: mismatch between use of anglo-american and metric units
multiple process errors: lack of formal interfaces

Mars Polar Lander, 1998
landing gear prematurely activated on entry to atmosphere
shock was interpreted as landing, engins were stopped, lander fell
error: lack of integration testing

Errors are manyfold, consequences may be grave

Interesting read:
Forum on Risks to the Public in Computers and Related Systems
`http://catless.ncl.ac.uk/Risks`

Errors causes and costs

Error dynamics in software development [John Rushby, SRI]

– 20-50 errors/kloc before testing; 2-4 errors/kloc after
– formal code inspection can reduce before-testing errors 10-fold !

Case study on 10kloc distributed real-time code:
– verification and validation: 52% cost (57% time)
– of this, 27% cost in inspection, 73% in testing
– 21% due to 4 defects uncovered in final testing
    (one of these originated in design phase)
– error elimination in detailed code inspection: 160 times more efficient than in testing !

Error causes and costs (cont.)

NASA JPL (Voyager and Galileo probes)

– majority: deficiencies in requirement and interface specification
– 1 error in 3 pages of requirements and 21 pages of code
– only 1 in 3 were programming errors
– 2/3 of functional errors: omissions in requirement specifications
– majority of interface errors: due to bad communication

Verification and validation: terminology

Software Engineering Institute – Capability Maturity Model Integration:

Verification ensures the product is built in accordance with requirements, specifications and standards.

Are the stated specifications met?
(*are we building the product right?*)

Validation ensures the produce will be usable on the market

Does the product cover operational needs?
Can the product be used in the intended environment?
(*are we building the right product?*)

V & V: terminology

NASA Software Assurance Guidebook and Standard:

Software Verification and Validation (V&V): process of ensuring that software being developed or changed will satisfy functional and other requirements (validation)
and each step in the process of building the software yields the right products (verification).

“The differences between verification and validation are unimportant except to the theorist; practitioners use the term V&V to refer to all of the activities that are aimed at making sure the software will function as required.”

V&V: process and technologies

Tehnologies

– inspection
  review (analysis by a third party)
  inspection (as above but with rigorously defined process and roles)
  walkthroughs (presentation soliciting opinions)
– testing
– formal verification and analysis

Process

– concept phase: designing a test plan
– req. analysis: test scenarios based on use cases
– design: verifying model wrt specification
– implementation: inspection + unit testing
– integration: integration testing, test reports

What is testing ?

“Testing is the process of executing a program with the intent of finding errors” (Myers, The Art of Software Testing).

What is testing ?

"Testing is the process of executing a program with the intent of finding errors" (Myers, The Art of Software Testing).

Sounds similar, but isn't (false, actually)
– "Testing is the process by which one shows the program does not have errors".
(impossible just through testing)

Dijkstra: "Testing can be used very effectively to show the presence of bugs but never to show their absence."

$\Rightarrow$ a *succesful* test discovers (and localizes) an error

What is a tester ?

The role of a tester is *to find errors*

- as early as possible
  (repair cost increases with time)
- and ensure they get fixed
  (reports, debugging, maintenance)

(Patton, Software Testing)

What Makes a Good Software Tester (Patton)

They are explorers.
They are troubleshooters
They are relentless
They are creative
They are (mellowed) perfectionists
They exercise good judgment
They are tactful and diplomatic (?)
They are persuasive (!)

Testing principles (Myers)

A test case must define the expected result or output.
– otherwise, we will see what we want to see

A programmer should avoid testing their own program
– psychologically, does not want to find errors
– exception: uni testing / test-driven development
Corollary: test group should not be development group

We need test cases for valid and invalid inputs.
Must test program does what is needed and doesn't do what it shouldn't.

Keep and reuse test cases!

Don't plan the test process assuming there won't be errors!

Probability to find errors in a piece of code is proportional to number of errors already found.

Testing “Axioms” (Patton)

Software testing is an exercise of evaluating risks

The more errors you find, the more there still are

Pesticide paradox (Beizer): errors become resilient to tests
(to find new errors, one needs new tests)

Not all errors found will be corrected

Product specifications are never definitive

Testers are not the most popular project team members :)

Software testing is a technical profession governed by a discipline

Testing discipline: organizations and metrics

Sample brief test report [Marnie Hutcheson, Software Testing Fundamentals]

"As per our agreement, we have tested 67 percent of the test inventory [...] the most important tests in the inventory as determined by our joint risk analysis.

The bug find rates and the severity composition of the bugs we found were within the expected range. Our bug fix rate is 85 percent.

It has been three weeks since we found a Severity 1 issue. There are currently no known Severity 1 issues open. Fixes for the last Severity 2 issues were regression-tested and approved a week ago. Overall, the system seems to be stable.

The load testing has been concluded. The system failed at 90 percent of the design load. The system engineers [...] will need 3 months to implement the fix.

Our recommendation is to ship on schedule, with the understanding that we have an exposure if the system utilization exceeds the projections before we have a chance to install the previously noted fix."

Testing – fundamental questions

[Cem Kaner, Black-box software testing course, Florida Inst. of Tech]

What do we test ? What do we want to achieve ?
What is the testing mission ?

How do we organize work to achieve the mission ?
The test strategy problem

When have we tested enough ?
The problem of measurement in testing

Testing – a more general vision

“A technical investigation conducted to provide quality-related information about a software product to a stakeholder” [Kaner]

investigation: *active, organized* search for information

technical: experiments, logic, models, algorithms, tools

software product: everything the client gets
(software, hardware, databases, documentation, etc.)

stakeholders: in success of product, and of testing

Testing purpose

[Kaner 2003 – What is a good test case ?]
– Find defects: especially in interesting parts (good coverage)
– Maximize bug count: in limited time
– Block premature release + help make ship/no-ship decision
– Minimize technical support cost
– Assess conformance to specifications / rules / standards
– Minimize risk (incl. safety-related lawsuit risk)
– Find scenarios where product works (despite bugs) – workarounds
– Assess quality: but: cannot *ensure* quality just by testing

What can testing not do ?
– verify correctness (absence of errors)
– ensure quality (QA is a process issue)

What is a good test case? (Kaner)

powerful: high chance of discovering bug if present

credible/realistic (to stakeholders): no corner cases (except: safety!)

representative / likely to be encountered by customer

easy to evaluate (is it a bug or not?) / easy to debug / informative

appropriately complex (progressive)

offer insight into some aspect of product / customer / environment
(e.g. detect change in behavior / performance)