Introduction to Software Testing

(*2nd edition*)

Chapter 1

What is Software Testing? Why Do We Test Software?

Instructor: **Morteza Zakeri**

Slides by: Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Modified by: **Morteza Zakeri**

Updated August 2018

First version, 28 August 2011

Lecturer

- **Morteza Zakeri**
  - Ph.D. in Computer Engineering (Software),
  - Iran University of Science and Technology (IUST)
  - *https://m-zakeri.github.io*
- Member of
  - Software Reverse Engineering Research Laboratory
    - *http://reverse.iust.ac.ir*
  - Association for Computing Machinery (ACM)
    - https://member.acm.org/~mzakeri-nasrabadi
- Interested in
  - Program analysis and transformation
  - Software testing and software quality assurance
  - Artificial intelligence for software engineering (AI4SE)
  - Compilers

About the IUST

- The university is among the top 4 university in Iran
- Located at Tehran city (Narmak)

Testing in the 21st Century

- Software defines behavior
  - network routers, finance, switching networks, other infrastructure
- Today's software market:
  - is much bigger
  - is more competitive
  - has more users

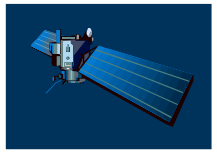
Industry is going through a revolution in what testing means to the success of software products

- Embedded Control Applications
  - airplanes, air traffic control
  - spaceships
  - watches
  - ovens
  - remote controllers
  - PDAs
  - memory seats
  - DVD players
  - garage door openers
  - cell phones
- Agile processes put increased pressure on **testers**
  - Programmers must unit test – with no training or education!
  - Tests are key to functional requirements – but who builds those tests?

Software is a Skin that Surrounds Our Civilization

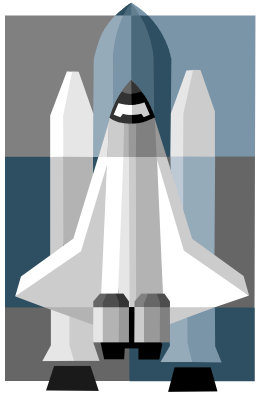
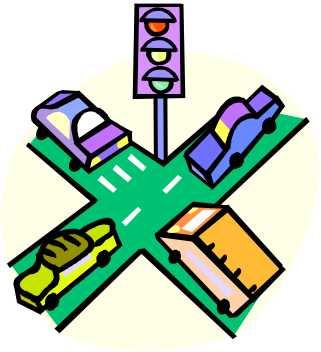










Quote due to Dr. Mark Harman

Software Faults, Errors & Failures

- **Software Fault:** A static defect in the software
- **Software Error:** An incorrect internal state that is the manifestation of some fault
- **Software Failure:** External, incorrect behavior with respect to the requirements or other description of the expected behavior

Faults in software are equivalent to design mistakes in hardware.
Software does not degrade.

Fault and Failure Example

- A patient gives a doctor a list of symptoms
  - Failures
- The doctor tries to diagnose the root cause, the ailment
  - Fault
- The doctor may look for anomalous internal conditions (high blood pressure, irregular heartbeat, bacteria in the blood stream)
  - Errors

Most medical problems result from external attacks (bacteria, viruses) or physical degradation as we age.

Software faults were there at the beginning and do not "appear" when a part wears out.

A Concrete Example

Fault: Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] == 0)
      {
         count++;
      }
   }
   return count;
}
```

Test 1
[2, 7, 0]
Expected: 1
Actual: 1

Error: i is 1, not 0, on the first iteration
Failure: none

Test 2
[0, 2, 7]
Expected: 1
Actual: 0

Error: i is 1, not 0
Error propagates to the variable count
Failure: count is 0 at the return statement

The Term Bug

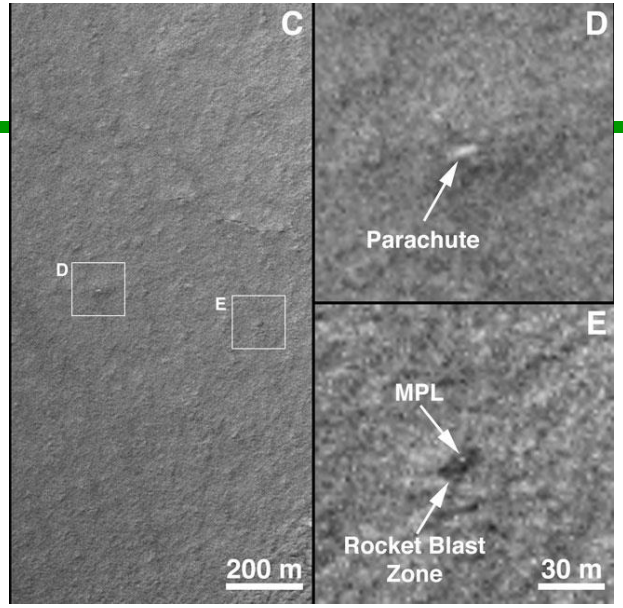
- *Bug* is used informally
- Sometimes speakers mean fault, sometimes error, sometimes failure … often the speaker doesn't know what it means !
- This class will try to use words that have precise, defined, and unambiguous meanings

"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and *[it is]* then that 'Bugs'—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite. . ." – Thomas Edison

"an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders." – Ada, Lovelace (notes on Babbage's Analytical Engine)

Spectacular Software Failures

- NASA's Mars lander: September 1999, crashed due to a units integration fault



Mars Polar Lander crash site?

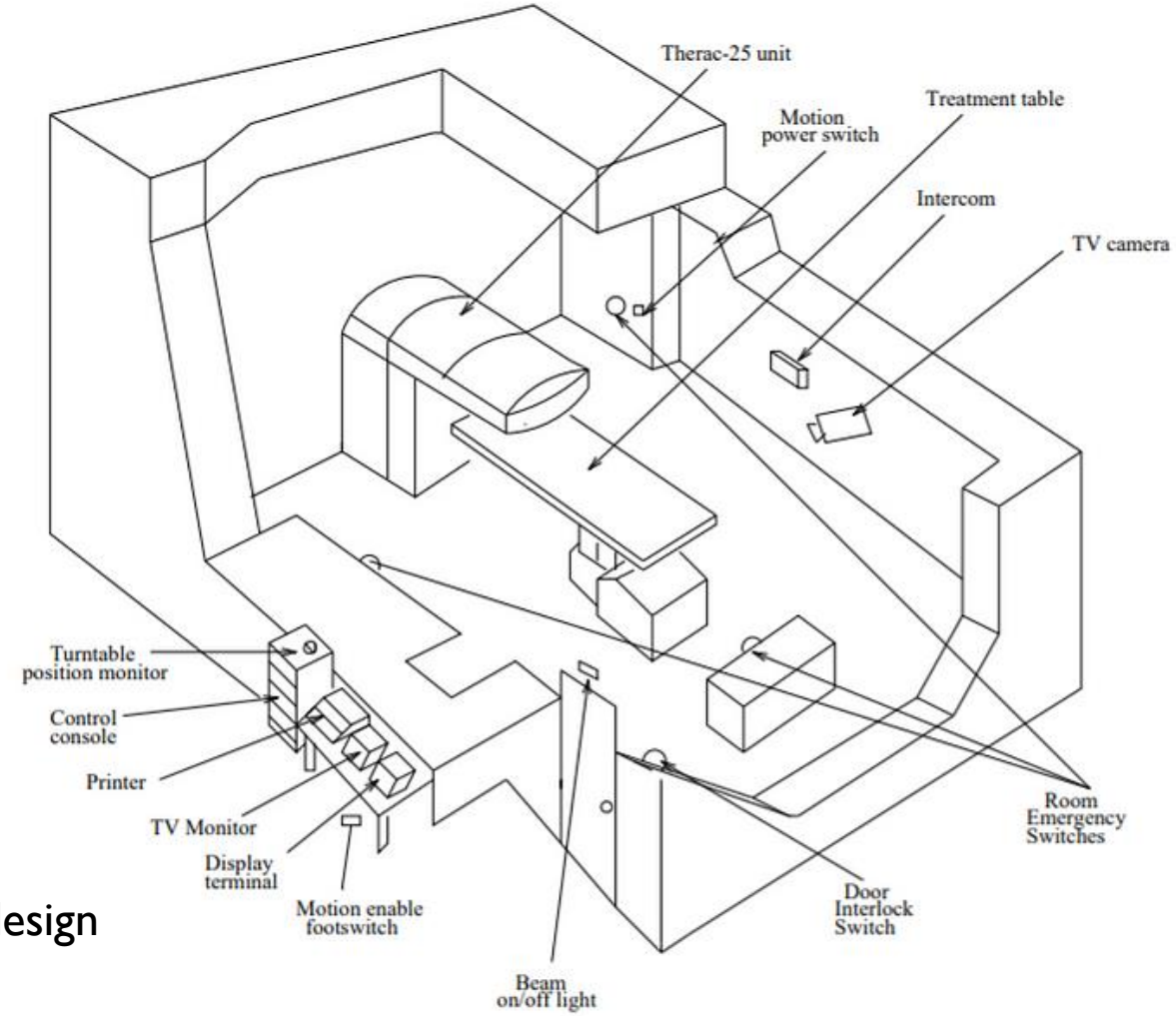
- Ariane 5 explosion : Millions of $$
- Intel's Pentium FDIV fault : Public relations nightmare

We need our software to be *dependable*

Testing is *one* way to assess dependability

Spectacular Software Failures (Cont'd)

- THERAC-25 radiation machine : Poor testing of safety-critical software can cost *lives* : **3** patients were killed



THERAC-25 design

Northeast Blackout of 2003

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

Financial losses of $6 Billion USD

The alarm system in the energy management system failed due to a software error and operators were not informed of the power overload in the system.

Costly Software Failures

- NIST report, "The Economic Impacts of Inadequate Infrastructure for Software Testing" (2002)
  - Inadequate software testing costs the US alone between $22 and $59 billion annually
  - Better approaches could cut this amount in half
- Huge losses due to web application failures
  - Financial services : $6.5 million per hour (just in USA!)
  - Credit card sales applications : $2.4 million per hour (in USA)
- In Dec 2006, *amazon.com's* BOGO offer turned into a double discount
- 2007 : Symantec says that most security vulnerabilities are due to faulty software

World-wide monetary loss due to poor software is staggering

Spectacular software failures

- Boeing A220: Engines failed after software update allowed excessive vibrations

- Boeing 737 Max: Crashed due to overly aggressive software flight overrides (MCAS)
- Toyota brakes: Dozens dead, thousands of crashes

- Healthcare website: Crashed repeatedly on launch—never load tested
- Northeast blackout: 50 million people, $6 billion USD lost … alarm system failed

Software testers try to find faults before the faults find users

Testing in the 21st Century

- More safety critical, real-time software
- Embedded software is ubiquitous … check your pockets
- Enterprise applications means bigger programs, more users
- Paradoxically, free software increases our expectations !
- Security is now all about software faults
  - Secure software is reliable software
- The web offers a new deployment platform
  - Very competitive and very available to more users
  - Web apps are distributed
  - Web apps must be highly reliable

The industry desperately needs our inventions!

The True Cost of Software Failure

Fail watch analyzed news articles for 2016

- 606 reported software failures
- Impacted half the world's population
- Cost a combined $1.7 trillion US dollars

Poor software is a significant drag on the world's economy

Not to mention frustrating

What Does This Mean?

Software testing is getting more important …

What are we trying to do when we test?

What are our goals?

Validation & Verification (*IEEE*)

- Validation: The process of evaluating software at the end of software development to ensure compliance with intended usage.

- Verification: The process of determining whether the products of a given phase of the software development process fulfill the requirements established during the previous phase.

IV&V stands for "*independent verification and validation*"

Testing Goals Based on Test Process Maturity

- **Level 0**: There's no difference between testing and debugging
- **Level 1**: The purpose of testing is to show correctness
- **Level 2**: The purpose of testing is to show that the software doesn't work.
- **Level 3**: The purpose of testing is not to prove anything specific, but to reduce the risk of using the software.
- **Level 4**: Testing is a mental discipline that helps all IT professionals develop higher quality software.

Level 0 Thinking

- Testing is the same as **debugging**
- Does not distinguish between incorrect **behavior** and mistakes in the program
- Does not help develop software that is **reliable** or **safe**

This is what we teach undergraduate CS majors

Level 1 Thinking

- Purpose is to show correctness
- Correctness is impossible to achieve
- What do we know if no failures?
  - Good software or bad tests?
- Test engineers have no:
  - Strict goal
  - Real stopping rule
  - Formal test technique
  - Test managers are powerless

This is what hardware engineers often expect

Level 2 Thinking

- Purpose is to show failures
- Looking for failures is a negative activity
- Puts testers and developers into an adversarial relationship
- What if there are no failures?

This describes most software companies.

How can we move to a *team approach*??

Level 3 Thinking

- Testing can only show the presence of failures
- Whenever we use software, we incur some risk
- Risk may be small and consequences unimportant
- Risk may be great and consequences catastrophic
- Testers and developers cooperate to reduce risk

This describes a few "enlightened" software companies

Level 4 Thinking

A mental discipline that increases quality

- Testing is only one way to increase quality
- Test engineers can become technical leaders of the project
- Primary responsibility to measure and improve software quality
- Their expertise should help the developers

This is the way "traditional" engineering works

Where Are You?

Are you at level 0, 1, or 2 ?

Is your organization at work at level 0, 1, or 2 ?

Or 3?

We hope to teach you to become "change agents" in your workplace …

Advocates for level 4 thinking

Tactical Goals : Why Each Test ?

If you don't know why you're conducting each test, it won't be very helpful

- Written test objectives and requirements must be documented
- What are your planned coverage levels?
- How much testing is enough?
- Common objective – spend the budget … test until the ship-date …
  - Sometimes called the "date criterion"

Here! Test This!

Offutt's first "professional" job

A stack of computer printouts—and no documentation

Why Each Test?

If you don't start planning for each test when the functional requirements are formed, you'll never know why you're conducting the test

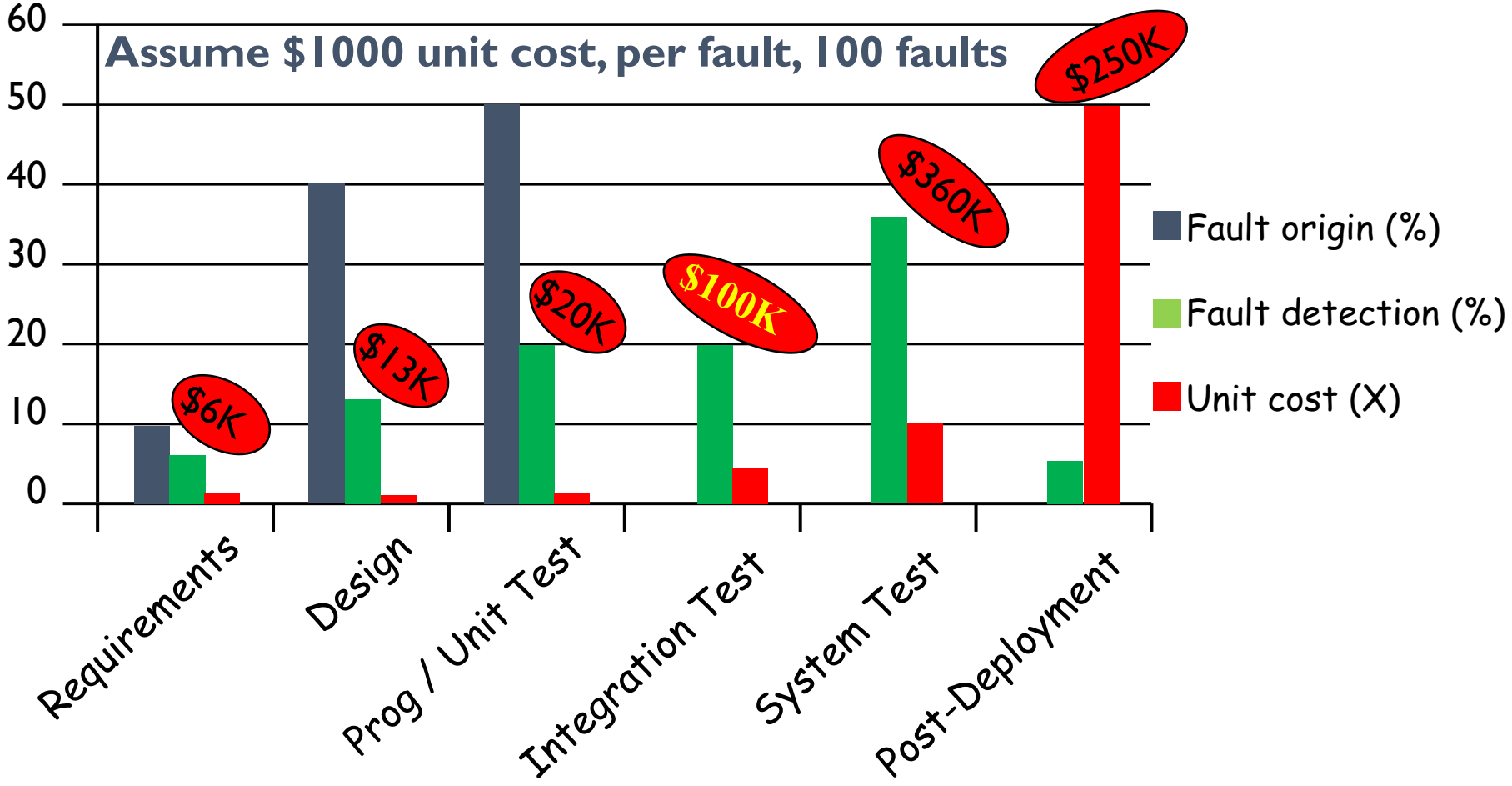
- 1980: "The software shall be easily maintainable"
- Threshold reliability requirements?
- What fact does each test try to verify?
- Requirements definition teams need testers!

Cost of Not Testing

Poor Program Managers might say:
"Testing is too expensive."

- Testing is the **most time consuming** and expensive part of software development
- Not testing is even more expensive
- If we have too little testing effort early, the cost of testing increases
- Planning for testing after development is prohibitively expensive

Cost of Late Testing



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

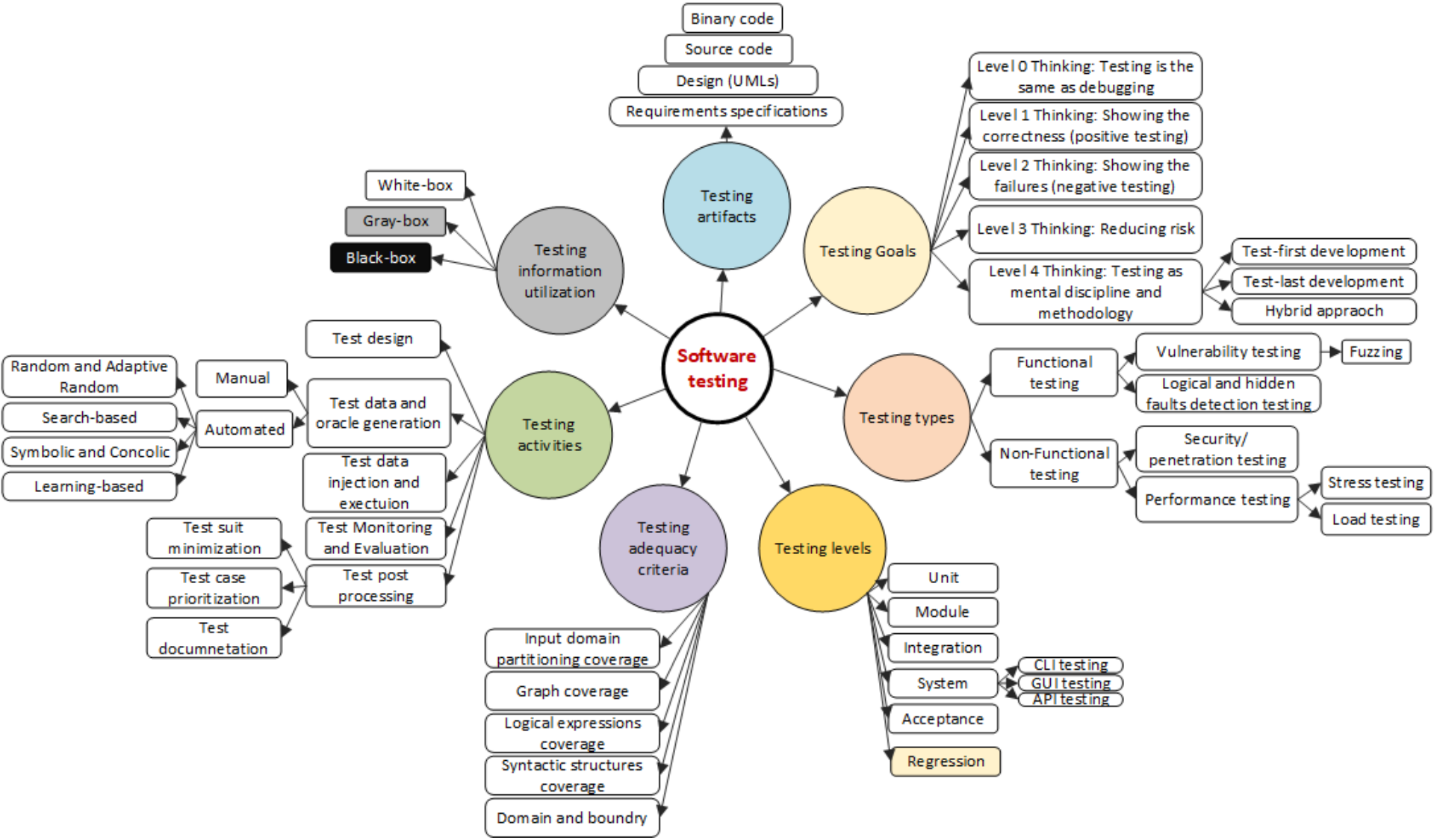
Summary: Why Do We Test Software?

A tester's goal is to eliminate faults as early as possible

- Improve quality
- Reduce cost
- Preserve customer satisfaction

Software Testing Taxonomy and Classification

- My proposed classification for software testing (version 1.0)



This course

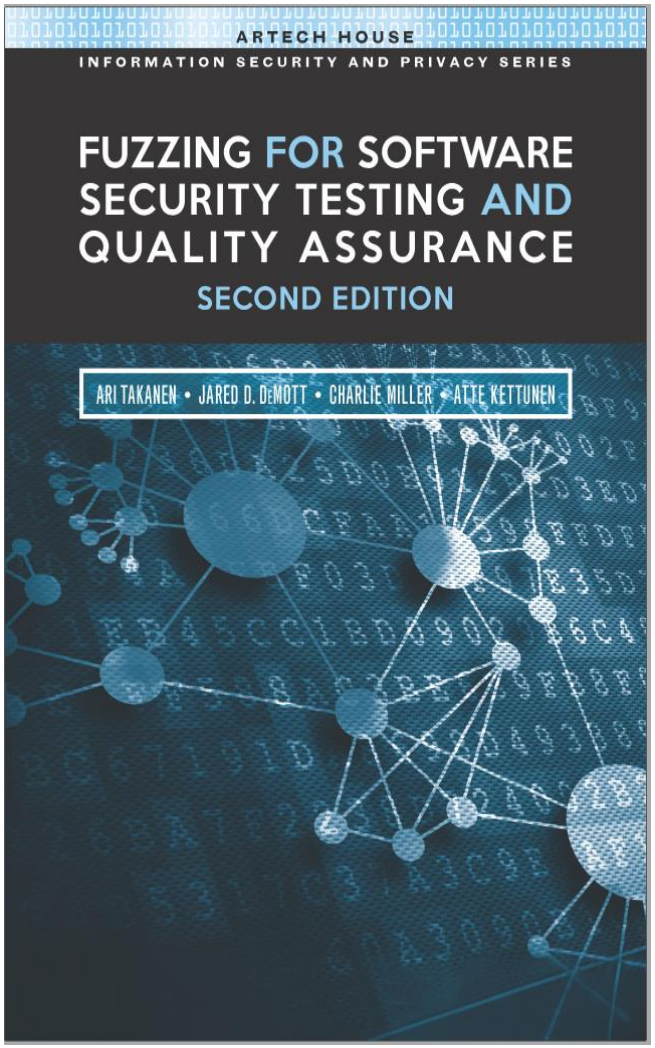
Lecture	Topic	Tools
L01-L04	Introduction: Why test, Type of tests, Test automation, and Agile test	JUnit, NUnit
L05-L09	Functional testing (Verification), Criteria-Based Testing	Graph Coverage, Data Flow Coverage, Domain Coverage, CodA
L10	Automated test generation (Symbolic Execution, Concolic Execution, and Taint analysis, Search-based testing)	CREST, CROWN, KLEE, JDART, Triton, EvoSuite, Randoop
L11	GUI (Desktop and Web Applications) testing	Selenium, Katalon Studio, GUITAR,
L12	Performance testing (Load, Stress)	JMeter
L13	Security (penetration) testing	Burp Suite, Acunetix
L14	Negative testing (Fuzzing)	AFL, Peach, FileFuzz, DeepFuzz

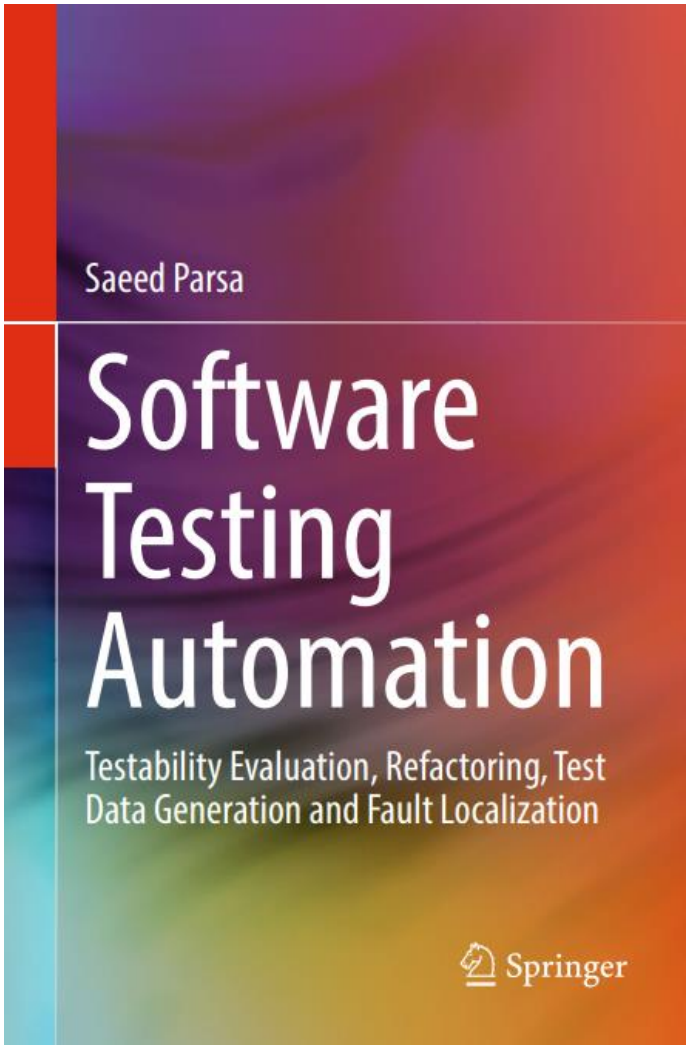
References

- [1] Ammann, P., & Offutt, J. (2016). **Introduction to software testing**. Cambridge University Press. https://doi.org/DOI: 10.1017/9781316771273

References

- [2], [3]



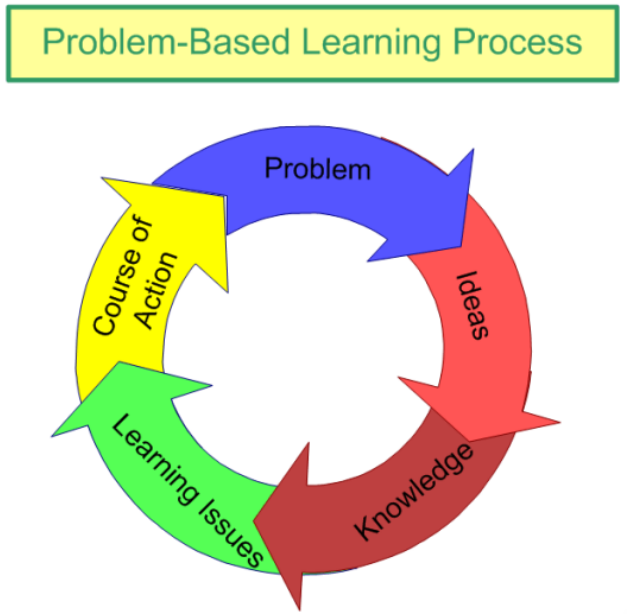


Grading policy

- Exams (50% or 10 out of 20)
- HWs (20% or 4 out of 20)
- Presentations --- tool demonstration (20% or 4 out of 20)
  - Slide, Codes, Reports.
- Class activity (10% or 2 out of 20)
- Extra works ([-1, 2+))
  - (Course slides, questions with solutions)

Problem-based Learning

- Traditional Teaching
  - Professor → slides/Blackboard
  - Teaching Assistance → Solve practical
- Problem Based Learning (PBL)
  - More Student-Centric
  - Professor proposes a problem
  - Students explore the solution space
  - Conditions:
    - Number of students



Stay hungry stay foolish