The following are modified versions of the publicly-available slides for Chapters 1 and 2 in the Ammann and Offutt Book, “Introduction to Software Testing” (http://www.cs.gmu.edu/~offutt/softwaretest)
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
Software Faults, Errors & Failures

- **Software Fault**: A static defect in the software

- **Software Failure**: External, incorrect behavior with respect to the requirements or other description of the expected behavior

- **Software Error**: An incorrect internal state that is the manifestation of some fault
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;
}
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: when needed, we will use the more precise terminology

---

“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 *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, Countess Lovelace

(notes on Babbage’s Analytical Engine)
Outline for today’s class

- Fundamental testing terminology ✓
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
Spectacular Software Failures

- NASA’s Mars lander: September 1999, crashed due to a units integration fault.
- THERAC-25 radiation machine: Poor testing of safety-critical software can cost lives: 3 patients were killed.
- 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.

Ariane 5: exception-handling bug: forced self destruct on maiden flight (64-bit to 16-bit conversion: about 370 million $ lost)

THERAC-25 design

Mars Polar Lander crash site?
Northeast Blackout of 2003

- 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.
More recent 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

Software testers try to find faults before the faults find users
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
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

Assume $1000 unit cost, per fault, 100 faults

Fault origin (%)
Fault detection (%)
Unit cost (X)

Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002
Software testing is getting more important

What are we trying to do when we test?
What are our goals?
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
Testing Goals Based on Test Process Maturity

- **Level 0**: There’s no difference between testing & 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 level is usually taught in 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?

- **Software engineers have no:**
  - Strict goal
  - Real stopping rule
  - Formal test technique
  - Test managers are powerless
Level 2 Thinking

- Purpose is to show failures

- Looking for failures can be a negative activity

- It can put testers and developers into an adversarial relationship

- What if there are no failures?

  This describes many software organizations.

  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 relatively few “enlightened” software organizations
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
What testing level are you at?

In-class activity: Poll

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

Advocates for level 4 thinking
What should testers aim for?

A tester should aim to eliminate faults as early as possible

• Improve quality
• Reduce cost of finding bugs
• Preserve customer satisfaction
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
A key Software Testing Limitation

Testing can only show the presence of failures

Not their absence
Moving beyond Level 0: Testing vs. Debugging

- **Testing**: Evaluating software by observing its execution

- **Test Failure**: Execution of a test that results in a software failure

- **Debugging**: The process of finding a fault given a failure

Not all inputs will “trigger” a fault into causing a failure
Four conditions necessary for a failure to be observed

1. **Reachability**: The location or locations in the program that contain the fault must be reached.

2. **Infection**: The state of the program must be incorrect.

3. **Propagation**: The infected state must cause some output or final state of the program to be incorrect.

4. **Reveal**: The tester must observe part of the incorrect portion of the program state.
RIPR Model

- **Reachability**
- **Infection**
- **Propagation**
- **Revealability**
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
Traditional Testing Levels

- **Acceptance testing**: Is the software acceptable to the user?
- **System testing**: Test the overall functionality of the system
- **Integration testing**: Test how modules interact with each other
- **Module testing (developer testing)**: Test each class, file, module, component
- **Unit testing (developer testing)**: Test each unit (method) individually

This view obscures underlying similarities
Object-Oriented Testing Levels

- **Inter-class testing**: Test multiple classes together

- **Inter-method testing**: Test pairs of methods in the same class

- **Intra-class testing**: Test an entire class as sequences of calls

- **Intra-method testing**: Test each method individually
Old View: Colored Boxes

- **Black-box testing**: Derive tests from external descriptions of the software, including specifications, requirements, and design.

- **White-box testing**: Derive tests from the source code internals of the software, specifically including branches, individual conditions, and statements.

- **Model-based testing**: Derive tests from a model of the software (such as a UML diagram).

**Model-Driven Test Design makes these distinctions less important.**

The more general question is: **from what abstraction level do we derive tests?**
Outline for today’s class

- Fundamental testing terminology
- The costs of insufficient, non-existent, or late testing
- The goals of a software tester
- Foundations of software testing
- Levels of software testing
- Types of testing activities
- Model-Driven Test Design
Testing can be broken up into four types of activities:

1. Test Design
   1.a) Criteria-based
2. Test Automation
   1.b) Human-based
3. Test Execution
4. Test Evaluation

Each type of activity requires different skills, background knowledge, education and training.

This class: you will learn something about each of these types of activities.
1. Test Design—(a) Criteria-Based

Design test values to satisfy coverage criteria or other engineering goal

- This is the most technical job in software testing

- Requires knowledge of:
  - Discrete math
  - Programming
  - Testing

- Assigning this task to people who are not trained to design tests is a sure way to get ineffective tests
1. Test Design—(b) Human-Based

Design test values based on domain knowledge of the program and human knowledge of testing

- This is much harder than it may seem to developers

- Criteria-based design can be blind to special situations

- Requires knowledge of:
  - Domain, testing, and user interfaces

- Requires almost no traditional CS
  - A background in the domain of the software is essential
2. Test Automation

Embed test values into executable scripts

- Often requires solutions to difficult problems related to observability and controllability

- Another challenge: how to determine, embed, and maintain the expected outputs?
3. Test Execution

Run tests on the software and record the results

- This is easy – and trivial if the tests are well automated

- These days, many organizations utilize a CI server for test execution
  - Travis
  - Jenkins
  - ??
Other Testing Activities

- **Test management**: Sets policy, organizes team, interfaces with development, chooses criteria, decides how much automation is needed, ...

- **Test maintenance**: Reuse tests as software evolves
  - Regression testing
  - Deciding when to trim the test suite is partly policy and partly technical – and very hard!

- **Test documentation**: Keep documentation in the automated tests
Outline for today’s class

- Fundamental testing terminology ✓
- The costs of insufficient, non-existent, or late testing ✓
- The goals of a software tester ✓
- Foundations of software testing ✓
- Levels of software testing ✓
- Types of testing activities ✓
- Model-Driven Test Design ➞ Next Lecture
Next Class

- Test Automation
  - JUnit
  - We may cover some Maven as well