Course Overview:

Verification and Validation

CSCE 747 - Lecture 1 - 01/10/2017

Today’s Goals

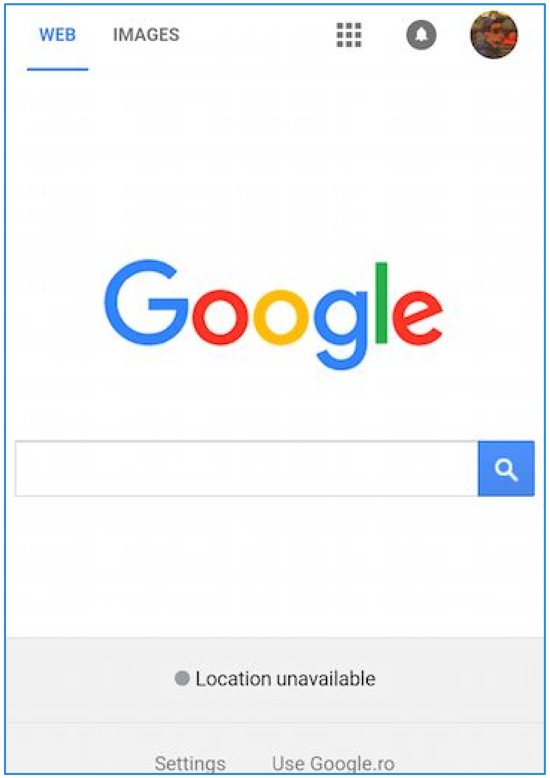
Introduce The Class

- AKA: What the heck is going on?
- Go over syllabus
- What you should already know
- Clarify course expectations
- Assignments/grading
- Answer any questions
- Cover the basics of verification and validation

When is software ready for release?

Our Society Depends on Software

This is software:



So is this:

Also, this:

Flawed Software Will Hurt Profits

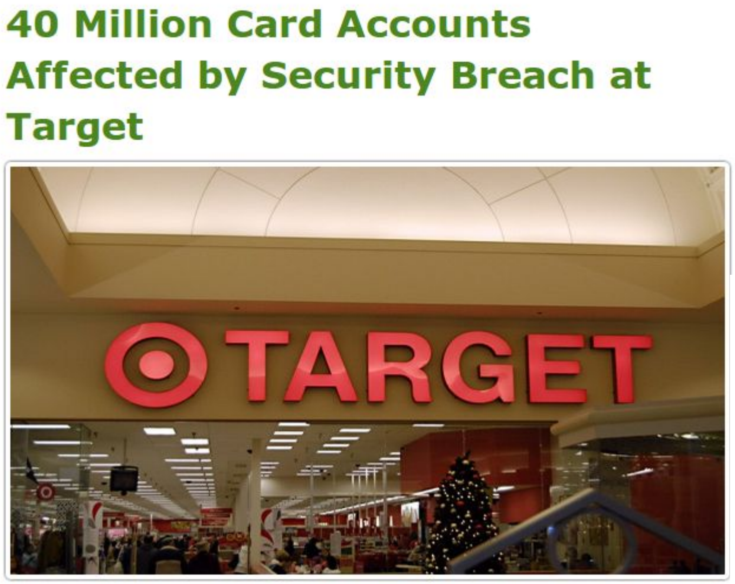
“Bugs cost the U.S. economy $60 billion annually… and testing would relieve one-third of the cost.”

- NIST

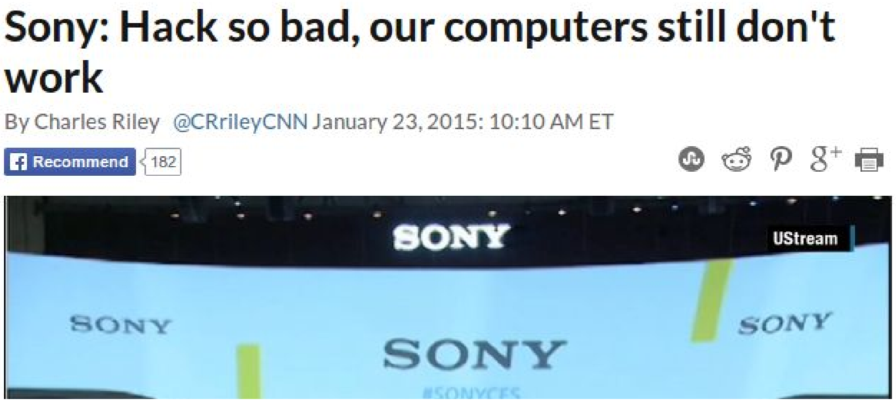
“Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it before.”

- Barry Boehm (TRW Emeritus Professor, USC)

Flawed Software Will Be Exploited

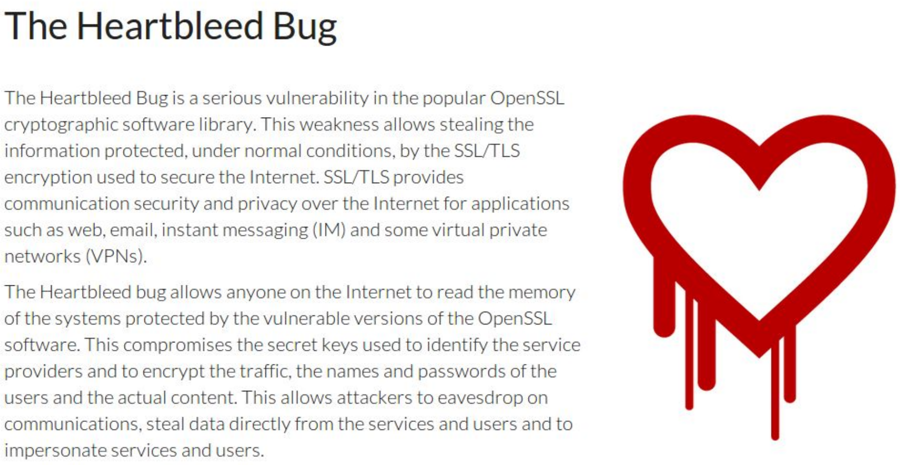


40 Million Card Accounts Affected by Security Breach at Target



Sony: Hack so bad, our computers still don't work

By Charles Riley @CRrileyCNN January 23, 2015: 10:10 AM ET



The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

Flawed Software Will Hurt People

In 2010, software problems were responsible for **26% of medical device recalls**.

“There is a reasonable probability that use of these products will cause serious adverse health consequences or death.”

- **US Food and Drug Administration**

This Course

- The key to good software?
  - **Verification and Validation**
    - Does the software do what we promised?
    - Does the software meet the needs of its users?
- In this course, we will explore the **testing** and **analysis** activities that make up the V&V process.

Desired Course Outcomes

1. The students will be familiar with the process of verification and validation.
2. … will understand the process of applying tests to software and the fundamental components of a test case.
3. … will be able to derive test cases from software requirement specifications - including being able to partition input and output domains, form test specifications, and identify valid combinations of input.
4. … will understand and be able to distinguish between methods of judging test case adequacy and how to design tests that will accomplish the obligations of such methods.
5. … will understand how to build models of system behavior and prove that their obey required properties.
6. … will be able to make logical arguments that prove the correctness of program implementations.
7. … will be able to write code to automate test execution and analysis.
8. … will be familiar with methods of measuring software reliability.

Lecture Plan (approximate)

- Introduction and Fundamentals (1 week)
- Functional and Combinatorial Testing (1 week)
- Test Case Adequacy/Structural Testing (1 week)
- Data Flow Testing (1 week)
- Testing Object-Oriented Software (1 week)
- Model-Based Testing/Finite State Verification (1 week)
- Proofs and Analysis (1 week)
- Execution and Automation (3 weeks)
- Inspections (1 week)
- End-of-Testing Activities (1 week)
- Presentations (2 week)

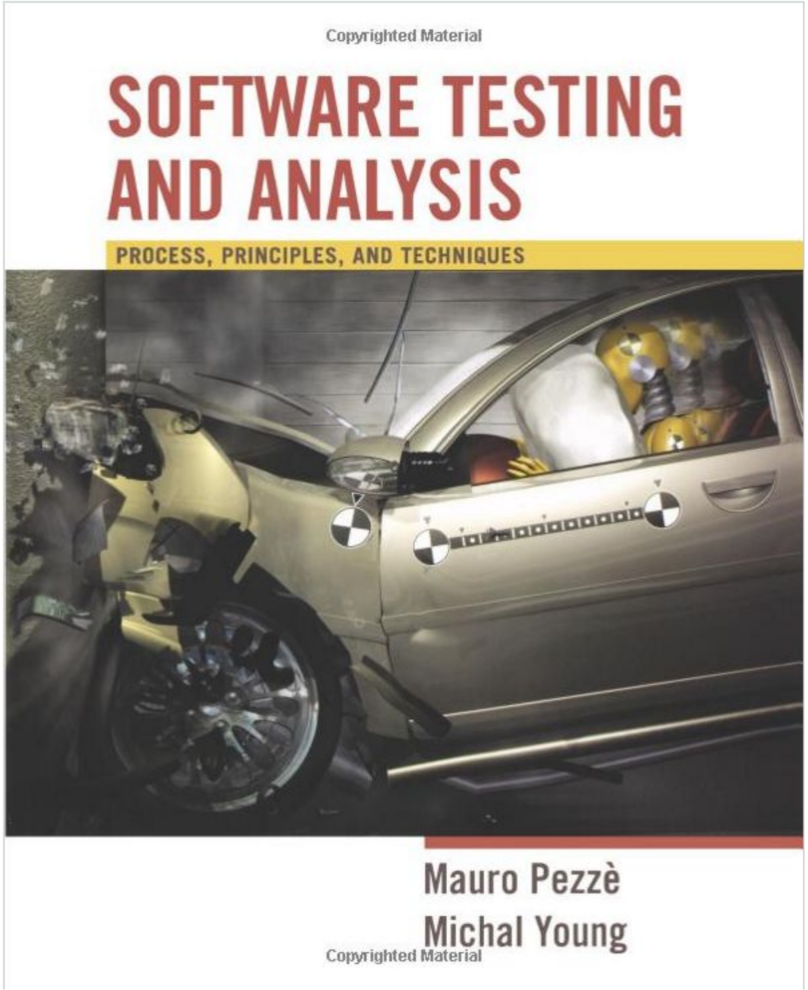
Contact Info

- Instructor: Greg Gay (Dr, Professor, $#*%)
  - E-mail: greg@greggay.com
  - Office Hours: T/Th, 4:00-5:00 PM, 3A66 Swearingen Engineering Center
- Website:
  - http://dropbox.cse.sc.edu/course/view.php?id=569
    - (Moodle - will be used for course material and assignment submission)
  - http://greggay.com/courses/spring17csce747/
    - (Static backup - somewhat behind, but useful if Moodle is down)

Textbook

Required:

- *Software Testing and Analysis*, Mauro Pezze and Michal Young.
  - (Only one edition)
  - Amazon: Rental for ~$25, Used for ~$40, new for ~$50
- Additional readings
  - 4-5 over the semester
  - Will be made available on Moodle

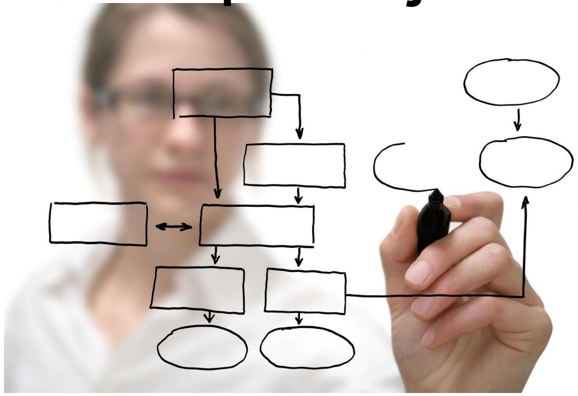

Copyrighted Material
SOFTWARE TESTING AND ANALYSIS
PROCESS, PRINCIPLES, AND TECHNIQUES
Mauro Pezzè
Michal Young
Copyrighted Material

Learning Modes

Lectures/Textbook

Class Discussions

Group Project



Prerequisites

CSCE 740 - Software Engineering

- Not essential, but very helpful.

You need to be proficient in Java

- (and, ideally, C++)
- You should be able to read and write programs without additional instruction.
- This is **not** a programming language class.

You need a basic understanding of algorithms, logic, and sets.

Assignments and Grading

- Group Assignments (40% in total)
  - Groups of 3.
  - Frequent peer evaluations.
- Individual Assignments (10%)
  - Reading + 1 page summary
- Midterm/Final Exams (20% each)
- Participation (10%)
  - In-class activities.
  - Group participation.
  - Answering questions.

Expected Workload

This class can be time consuming.

- Understanding the material takes time.
- Project work requires team coordination.

Do not underestimate the project work.

- Good engineering is hard.
- Planning and scheduling your time is essential.
- Do NOT delay getting started.
- Appoint a team leader (and rotate the role)

Feedback

Problems with assignments, course questions, feedback?

- Contact me! I like feedback!

Problem with instructor

- Also contact me
- Contact CS front office

Other Policies

Integrity and Ethics:

The homework and programs you submit for this class must be entirely your own. If this policy is not absolutely clear, then please contact me. Any other collaboration of any type on any assignment is not permitted. It is your responsibility to protect your work from unauthorized access.

Classroom Climate:

All students are expected to behave as scholars at a leading institute of technology. This includes arriving on time, not talking during lecture (unless addressing the instructor), and not leaving the classroom before the end of lecture. Disruptive students will be warned and potentially dismissed from the classroom.

Other Policies

Make-Up and Late Homework

- Make-ups for graded activities may be arranged if your absence is caused by a documented illness or personal emergency.
- Homework assignments are due at the time noted on the assignment handout. Late work is not accepted without prior approval. Any assignment turned in after the due date will be considered late and will be subject to a penalty of 10% per day, including weekends and holidays.

Other Policies

Diversity

Students in this class are expected to respectfully work with all other students, regardless of gender, race, sexuality, religion, or any other protected criteria. There is a zero-tolerance policy for any student that discriminates against other students.

Special Needs

We will provide, on a flexible and individual basis, reasonable accommodations to students that have disabilities that may affect their ability to participate in course activities or to meet course requirements Students with disabilities should contact their instructor early in the semester to discuss their individual needs.

When is software ready for release?

Basic Answer...

Software is ready for release when you can argue that it is ***dependable*.**

- Correct, reliable, safe, and robust.
- The primary process of making software dependable (and providing evidence of dependability) is **Verification and Validation**.

Verification and Validation

Activities that must be performed to consider the software “done.”

- **Verification:** The process of proving that the software conforms to its specified functional and non-functional requirements.
- **Validation:** The process of proving that the software meets the customer’s true requirements, needs, and expectations.

Verification and Validation

Barry Boehm, inventor of the term “software engineering”, describes them as:

- **Verification:**
  - “Are we building the product right?”
- **Validation:**
  - “Are we building the right product?”

Verification

- Is the implementation consistent with its specification?
    - “Specification” and “implementation” are roles.
        - Source code and requirement specification.
        - Detailed design and high-level architecture.
        - Test oracle and requirement specification.
- Verification is an experiment.
    - Does the software work under conditions we set?
    - We can perform trials, evaluate the software, and provide evidence for verification.

Validation

- Does the product work in the real world?
  - Does the software fulfill the users' actual requirements?
- Not the same as conforming to a specification.
  - If we specify and implement all behaviors related to two buttons, we can achieve verification.
  - If the user expected a third button, we have not achieved validation.

Verification and Validation

- Verification
  - Does the software work as intended?
- Validation
  - Does the software meet the needs of your users?
  - **This is much harder.**

Validation shows that software is useful. Verification shows that it is dependable. Both are needed to be ready for release.

Verification and Validation: Motivation

- Both are important.
  - A well-verified system might not meet the user’s needs.
  - A system can’t meet the user’s needs unless it is well-constructed.
- This semester largely focuses on verification.
  - How can we ensure that the software we build is dependable.
  - Testing is the primary activity of verification, and our main focus in this class.

Required Level of V&V

The goal of V&V is to establish confidence that the system is “fit for purpose.”

How confident do you need to be? Depends on:

- **Software Purpose:** The more critical the software, the more important that it is reliable.
- **User Expectations:** When a new system is installed, how willing are users to tolerate bugs because benefits outweigh cost of failure recovery.
- **Marketing Environment:** Must take into account competing products - features and cost - and speed to market.

Basic Questions

1. When do verification and validation start? When are they complete?
2. What techniques should be applied to obtain acceptable quality at an acceptable cost?
3. How can we assess readiness for release?
4. How can we control the quality of successive releases?
5. How can the development process be improved to make verification more effective (in cost and impact)?

When Does V&V Start?

- V&V starts as soon as the project starts.
- Feasibility studies must consider quality assessment.
- Requirement specifications can be used to derive test cases.
- Design can be verified against requirements.
- Code can be verified against design and requirements.
- Feedback can be sought from stakeholders at any time.

Types of Verification

Static Verification

- Analysis of static system artifacts to discover problems.
    - Proofs: Posing hypotheses and making a logical argument for their validity using specifications, system models, etc.
    - Inspections: Manual “sanity check” on artifacts (such as source code) by other people or tools, searching for issues.

Advantages of Static Verification

- During execution, errors can hide other errors. It can be hard to find all problems or trace back to a single source. Static inspections are not impacted by program interactions.
- Incomplete systems can be inspected without additional costs. If a program is incomplete, special code is needed to run the part that is to be tested.
- Inspection can also assess quality attributes such as maintainability, portability, poor programming, inefficiencies, etc.

Dynamic Verification

- Exercising and observing the system to argue that it meets the requirements.
  - Testing: Formulating controlled sets of input to demonstrate requirement satisfaction or find faults.
  - Fuzzing: Spamming the system with random input to locate security vulnerabilities, memory leaks, buffer overruns, etc.
  - Taint Analysis: Assigning a bad value to a variable and monitoring which system variables it corrupts and how it corrupts them.

Dynamic Verification

- Static verification is not good at discovering problems that arise from runtime interaction, timing problems, or performance issues.
- Dynamic verification is often cheaper than static - easier to automate.
  - However, it cannot prove that properties are met - cannot try all possible executions.

The Trade-Off Game

Software engineering is the process of designing, constructing and maintaining ***the best software possible*** given the ***available resources***.

We are always trading off between what we want, what we need, and what we've got. As a NASA engineer put it,

- **“Better, faster, or cheaper - pick any two”**

The Role of Software Engineers

Software engineers, therefore, aren't just responsible for designing, constructing, and maintaining software.

They are the people we look to ***plan***, ***make***, and ***justify well-informed decisions*** about ***trade-offs*** throughout the development process.

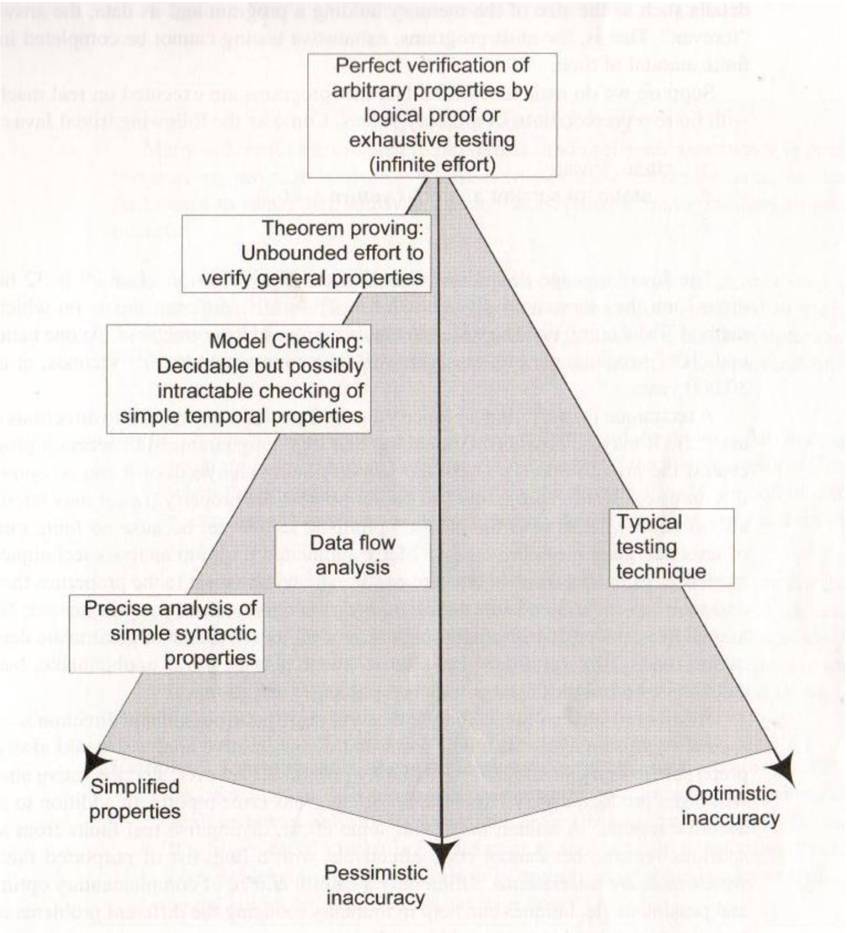
Perfect Verification

- For physical domains, verification consists of calculating proofs of correctness.
- Given a precise specification and a program, we should be able to do the same… Right?
    - Verification is an instance of the halting problem.
    - For each verification technique, there is at least one program for which the technique cannot obtain an answer in finite time.
        - Testing - cannot exhaustively try all inputs.
    - We must accept some degree of inaccuracy.

Verification Trade-Offs

Three dimensions of inaccuracy:

- **Pessimistic Inaccuracy** - not guaranteed to accept a program even if the program possesses the property.
- **Optimistic Inaccuracy** - may accept a program that does not possess a property.
- **Property Complexity** - if one property is too difficult to check, substitute one that is easier to check or constrain the types of programs checked.


Perfect verification of arbitrary properties by logical proof or exhaustive testing (infinite effort)
Theorem proving: Unbounded effort to verify general properties
Model Checking: Decidable but possibly intractable checking of simple temporal properties
Data flow analysis
Typical testing technique
Precise analysis of simple syntactic properties
Simplified properties
Optimistic inaccuracy
Pessimistic inaccuracy

Assessing Verification Techniques

- Safe
  - No optimistic inaccuracy - it only accepts programs that are correct with respect to that property.
- Sound
  - An analysis of a program with respect to property is *sound* if the technique returns true ONLY when the program does meet the property.
  - If true = correct and the technique is *sound,* then the technique is also *safe*.
  - If true = incorrect and the technique is sound, you allow *optimistic* but disallow *pessimistic inaccuracy*.

Assessing Verification Techniques

- Complete
  - An analysis of a property on a program is *complete* if it always returns true when the program does satisfy the program.
  - If true = correct, then *complete* admits only *optimistic inaccuracy*.
- Often a trade-off between safe, sound, and complete.

How Can We Assess the Readiness of a Product?

- Identifying faults is useful, but finding all faults is nearly impossible.
- Instead, need to decide when to stop verification and validation.
- Need to establish criteria for acceptance.
  - How good is “good enough”?
- One option is to measure dependability (availability, mean time between failures, etc) and set a “acceptability threshold”.

Product Readiness

- Another option is to put it in the hands of human users.
- Alpha/Beta Testing - invite a small group of users to start using the product, have them report feedback and faults. Use this to judge product readiness.
    - Can make use of dependability metrics for a quantitative judgement (metric > threshold).
    - Can make use of surveys as a qualitative judgement (are the users happy with the current product?)

Ensuring the Quality of Successive Releases

- Verification and validation do not end with the release of the software.
  - Software evolves - new features, environmental adaptations, bug fixes.
  - Need to test code, retest old code, track changes.
- Faults have not always been fixed before release. Do not forget those.
- Regression Testing - when code changes, rerun tests to ensure that it still works.
  - As faults are repaired, add tests that exposed them to the suite.

Improving the Development Process

- Try to learn from your mistakes in the next project.
- Collect data during development.
  - Fault information, bug reports, project metrics (complexity, # classes, # lines of code, coverage of tests, etc.).
- Classify faults into categories.
- Look for common mistakes.
- Learn how to avoid such mistakes.
- Share information within your organization.

We Have Learned

- Software should be dependable and useful before it is released into the world.
- Verification is the process of demonstrating that an implementation meets its specification.
  - This is the primary means of making software dependable (and demonstrating dependability).
  - Testing is the most common form of verification.

We Have Learned

- Verification techniques can be static or dynamic.
  - Often pessimistically or optimistically inaccurate
  - Level of inaccuracy can be controlled by simplifying properties.
  - Techniques strive to be safe, sound, and complete.
    - But, obtaining one often involves losing another.

Next Time

- More introduction:
  - Testing fundamentals.
  - Principles of analysis and testing.

- Reading:
  - Chapters 1-4 of testbook.
- Plan your team selection.
  - The earlier, the better!