Design Verification
COMS30026

WEEKLY STATUS UPDATE – W4

Kerstin Eder

Trustworthy Systems Lab

University of BRISTOL

Department of COMPUTER SCIENCE

Topics W1

- ✓ Introduction to DV
- ✓ Verification Hierarchy
- ✓ Fundamentals of Simulation-based Verification
  - Driving & Checking

- **Lab W1:**
  - get remote access to the EDA software
  - teach yourself Verilog ☺

Topics W2

- ✓Verification Tools
- ✓Hardware Design Languages
- ✓Verification Cycle, Methodology & Plan

- **Lab W2:**
  - Introduction to ModelSim/Questa
    - installed on linux lab machines
  - Work through mux testbench from Exercise 2

    https://uobdv.github.io/Design-Verification/

Topics W3

- ✓ Stimuli Generation Part I: Foundations
- ✓ Stimuli Generation Part II: Test Automation
- ✓ Checking

- **Lab W3:**
  - Introduction to **Practical 1** (Weeks 2-4)
    - Specification review
    - Debug calculator design using Modelsim/Questa
    - Example testbench on Blackboard

Topics W4

- ✓ Coverage Part I: Introduction and Code Coverage
- ✓ Coverage Part II: Functional Coverage
- Coverage Part III: Coverage Analysis

- **Lab W4:**

  Practical 1 (Weeks 2-4)

  – calc1 DUV provided as **black box design**
    - Specification review
    - Development of a **Verification Plan**
    - Debug calculator design using Modelsim/Questa
    - Example testbench on Blackboard

DESIGN VERIFICATION Practical 1:
Debugging a Calculator Design

This practical requires you to find bugs in a calculator design. The calculator specification is given below. The design HDL code is not available for this practical. You are expected to use a black box verification approach.

For simulation with ModelSim, the calculator design has been provided in the pre-compiled library `calc1_black_box`. The library is available from BlackBoard as a ZIP file. BlackBoard also provides you with instructions on how to set up your workspace with this library. The top-level module of the calculator design is called `calc1`. Please use the example testbench file provided on BlackBoard to check that you have set everything up correctly.

Calculator Specification

Input/Output Specification

The calculator has four commands: add, subtract, shift left and shift right. It can handle (but not process) four requests in parallel. All four requestors use separate input signals. All requestors have equal priority. Figure 1 shows the input output ports of the calculator.

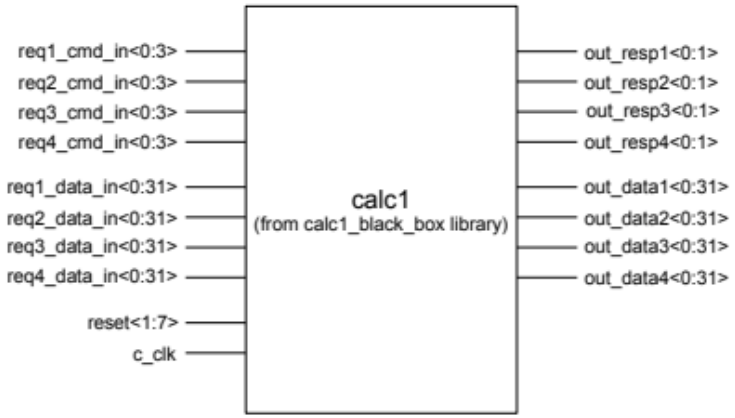


Figure 1: Top-level of Calculator Design

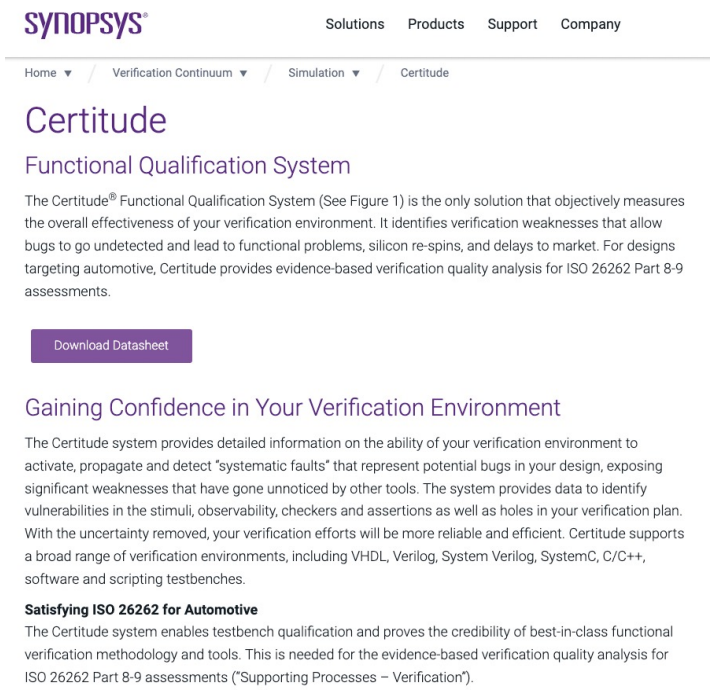
Opportunities – Week 4

- Review the paper on mutation testing and its automation

https://cs.gmu.edu/~offutt/rsrch/papers/mut00.pdf

- Investigate commercial tools such as Certitude

https://www.synopsys.com/verification/simulation/certitude.html



SYNOPSYS®

Solutions Products Support Company

Home ▾ / Verification Continuum ▾ / Simulation ▾ / Certitude

Certitude

Functional Qualification System

The Certitude® Functional Qualification System (See Figure 1) is the only solution that objectively measures the overall effectiveness of your verification environment. It identifies verification weaknesses that allow bugs to go undetected and lead to functional problems, silicon re-spins, and delays to market. For designs targeting automotive, Certitude provides evidence-based verification quality analysis for ISO 26262 Part 8-9 assessments.

Download Datasheet

Gaining Confidence in Your Verification Environment

The Certitude system provides detailed information on the ability of your verification environment to activate, propagate and detect "systematic faults" that represent potential bugs in your design, exposing significant weaknesses that have gone unnoticed by other tools. The system provides data to identify vulnerabilities in the stimuli, observability, checkers and assertions as well as holes in your verification plan. With the uncertainty removed, your verification efforts will be more reliable and efficient. Certitude supports a broad range of verification environments, including VHDL, Verilog, System Verilog, SystemC, C/C++, software and scripting testbenches.

Satisfying ISO 26262 for Automotive

The Certitude system enables testbench qualification and proves the credibility of best-in-class functional verification methodology and tools. This is needed for the evidence-based verification quality analysis for ISO 26262 Part 8-9 assessments ("Supporting Processes – Verification").

Mutation 2000: Uniting the Orthogonal*

A. Jefferson Offutt
ISE Department, 4A4
George Mason University
Fairfax, VA 22030-4444 USA
703-993-1654
ofut@ise.gmu.edu
www.ise.gmu.edu/faculty/ofut/

Roland H. Untch
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN 37132-0048
615-898-5047
untch@mtsu.edu
www.mtsu.edu/~untch/

Abstract

Mutation testing is a powerful, but computationally expensive, technique for unit testing software. This expense has prevented mutation from becoming widely used in practical situations, but recent engineering advances have given us techniques and algorithms for significantly reducing the cost of mutation testing. These techniques include a new algorithmic execution technique called schema-based mutation, an approximation technique called weak mutation, a reduction technique called selective mutation, heuristics for detecting equivalent mutants, and algorithms for automatic test data generation. This paper reviews experimentation with these advances and outlines a design for a system that will approximate mutation, but in a way that will be accessible to everyday programmers. We envision a system to which a programmer can submit a program unit and get back a set of input/output pairs that are guaranteed to form an effective test of the unit by being close to mutation adequate. We believe this system could be efficient enough to be adopted by leading-edge software developers. Full automation in unit testing has the potential to dramatically change the economic balance between testing and development, by reducing the cost of testing from the major part of the total development cost to a small fraction.

1. Introduction

Mutation analysis has a rich and varied history, with major advances in concepts, theory, technology, and social viewpoints. This history begins in 1971, when Richard Lipton proposed the initial concepts of mutation in a class term paper titled "Fault Diagnosis of Computer Programs." It was not until the end of the 1970's, however, before major work was published on the subject [1, 2, 3]; the DeMillo, Lipton, and Sayward paper [3] is generally cited as the seminal reference.

PIMS [1, 4, 5, 6], an early mutation testing tool, pioneered the general process typically used in mutation testing of creating mutants (of Fortran IV programs), accepting test cases from the users, and then executing the test cases on the mutants to decide how many mutants were killed. In 1987, this same process (of add test cases, run mutants, check results, and repeat) was adopted and extended in the Mothra mutation toolset [7, 8, 9, 10], which provided an integrated set of tools, each of which performed an individual, separate task to support mutation analysis and testing. Because each Mothra tool is a separate command, it was easy to incorporate, and thus experiment with, additional types of processing. Although a few other mutation testing tools have been developed since Mothra [11, 12, 13], Mothra is likely the most widely known mutation testing system extant.

Despite the relatively long history of mutation testing, the software development industry has failed to employ it. We posit that the three primary reasons why industry has failed to use mutation testing are the lack of economic incentives for stringent testing, inability to successfully integrate unit testing into software development processes, and difficulties with providing full and economical automated technology to support mutation analysis and testing. The first reason, the lack of economic incentives for applying highly advanced testing techniques, is beyond the scope of this paper, which is primarily technological in nature. On the other hand, software is increasingly being used to perform essential roles in applications that require high reliability, including safety-critical software (avionics, medical, and industrial control) infrastructure-critical software (telephony and networks), and commercial en-

*Supported by the National Science Foundation under awards CCR-9804011 and CCR-9707792.

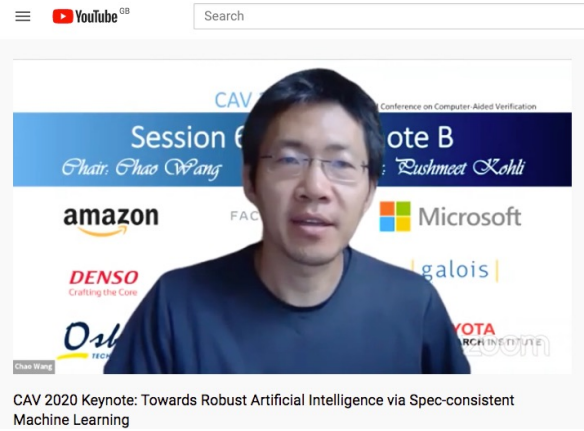
W. E. Wong (ed.), *Mutation Testing for the New Century*

Opportunities – Week 4

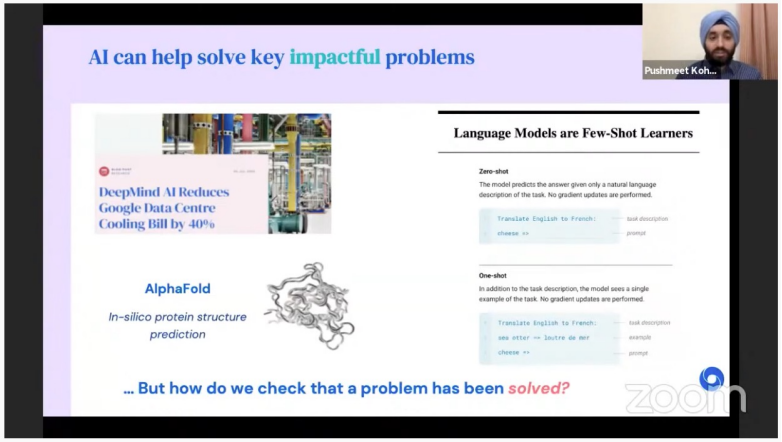
- **CAV 2020 Keynote:**

  Towards Robust Artificial Intelligence via Spec-consistent Machine Learning

  by Pushmeet Kohli from DeepMind

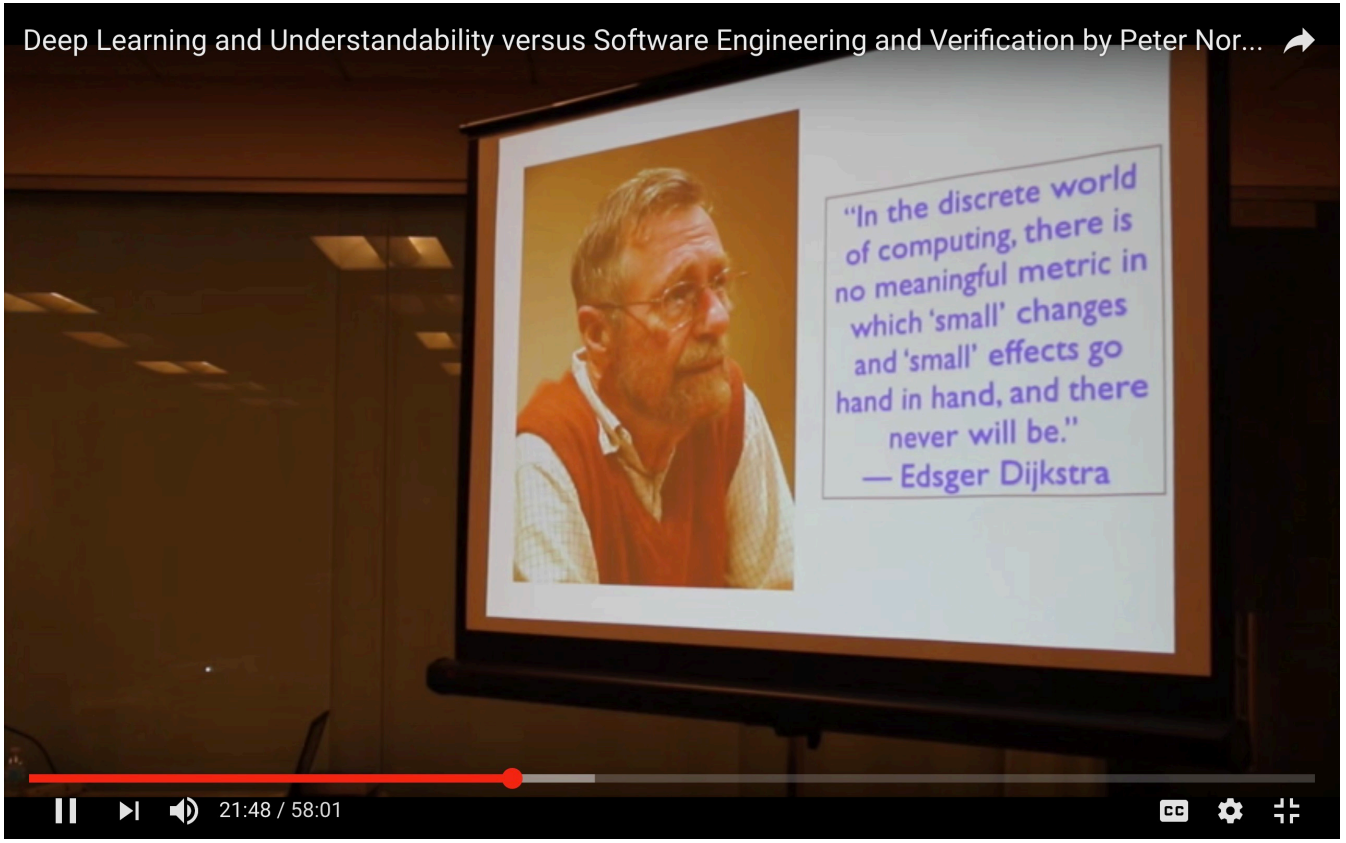


CAV 2020 Keynote: Towards Robust Artificial Intelligence via Spec-consistent Machine Learning



https://www.youtube.com/watch?v=25RhyyOg5Pg&feature=youtu.be

Opportunities – Week 4

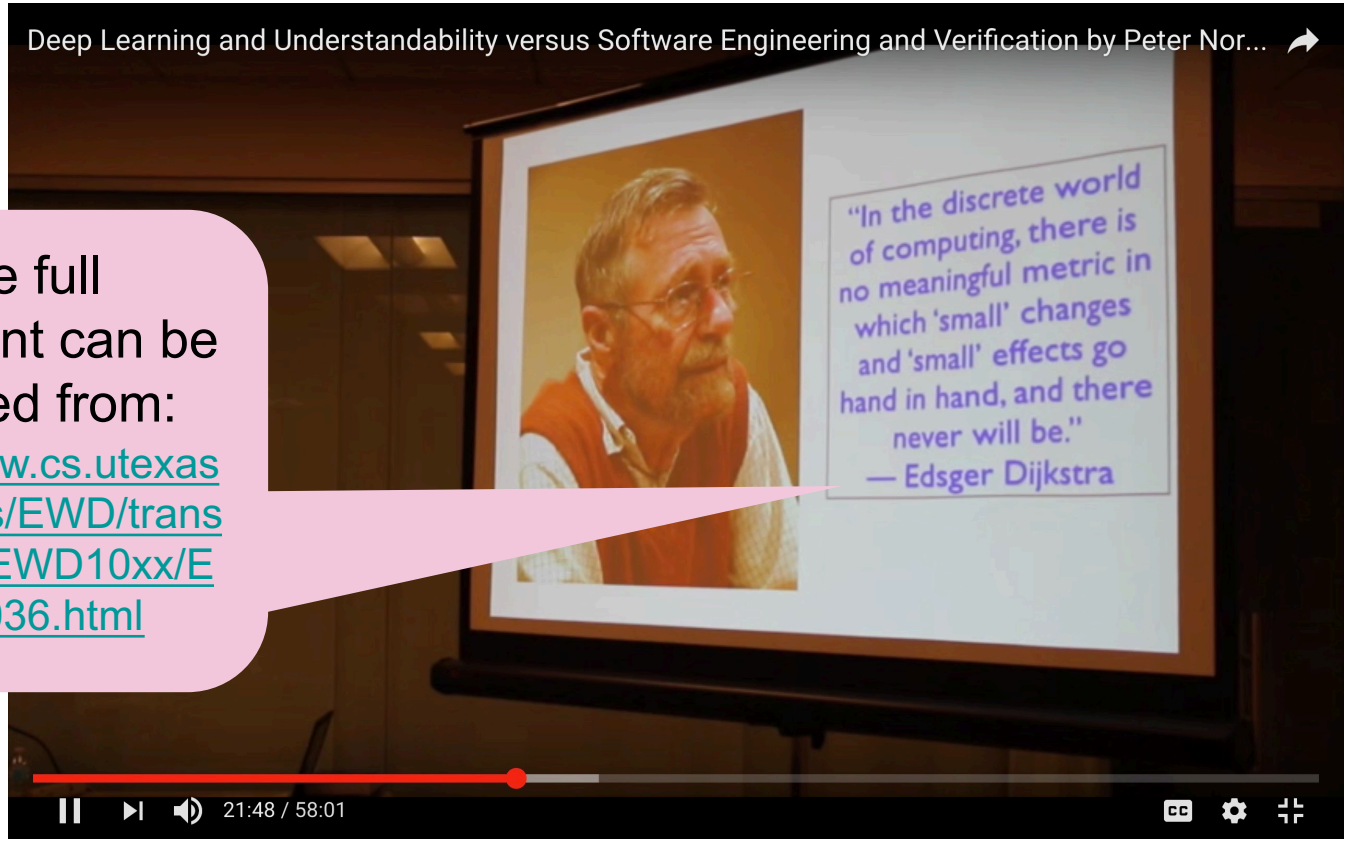


Peter Norvig on *Deep Learning and Understandability versus Software Engineering and Verification*

https://youtu.be/3Ulh8yUiuuY

Opportunities – Week 4

The full document can be sourced from: https://www.cs.utexas.edu/users/EWD/transcriptions/EWD10xx/EWD1036.html



Peter Norvig on *Deep Learning and Understandability versus Software Engineering and Verification*

https://youtu.be/3Ulh8yUiuuY

More fundamental questions

- Does testing work?
  - Is it valid to interpolate between test results?

Mechanical system ⟷ **Digital system**

The traditional testing assumption

- Mechanical scales behave continuously within limits.
- Allows interpolation between test results.

- With digital scales, the displayed weight depends on software.
- There is no underlying law to guarantee continuous behaviour.
- In general, it is **wrong** to interpolate between test results.

The traditional testing assumption

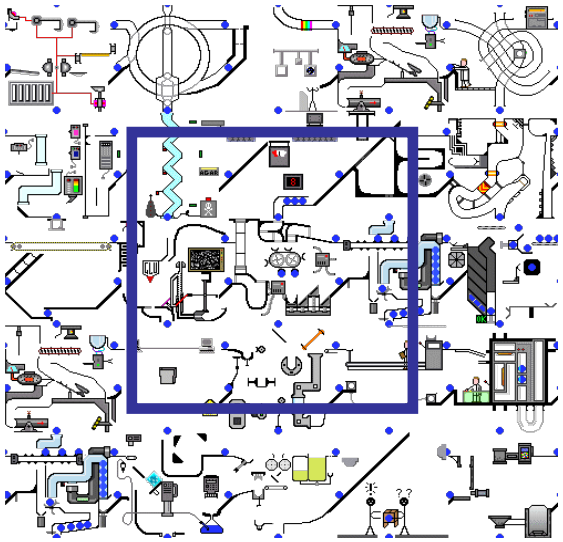
Most testing of digital systems is based on a false analogy!

- With digital scales, the displayed weight depends on software.
- There is no underlying law to guarantee continuous behaviour.
- In general, it is **wrong** to interpolate between test results.

More fundamental questions

Mechanical system ⟷ **Digital system**

- Does testing work?
  - Is it valid to interpolate between test results?
- Does testing scale?
  - It is impossible to test all interleavings, and impossible to test any meaningful fraction of system states.
- Is testing an appropriate method to gain confidence in the correctness of complex systems?
  - What does testing tell you?
    - Useful for validation
  - Are formal methods the answer?
    - Formal verification
    - Correctness by design
      - Can we ensure functional correctness?
      - Can we ensure safety?
      - Can we ensure no deadlocks?



Next

- Recordings of lectures for **Week 4:**
  - ✓ Coverage Part I: Introduction and Code Coverage
  - ✓ Coverage Part II: Functional Coverage
  - ✓ Coverage Part III: Coverage Analysis
  - uobdv.github.io/Design-Verification/ shows a **weekly schedule of topics** to watch BEFORE the next session, ideally
  - Recordings are available from Blackboard unit page
- Tasks for you this week:
  - Submit issues found during your **Specification review** to Blackboard Discussion Forum
  - Develop a **Verification Plan** for the calc1 black box DUV
  - **Attend the lab session** with Xuan to get help with using ModelSim/Questa so you can complete **Practical 1**

Questions

https://www.bristol.ac.uk/tsl