CS 4350: Fundamentals of Software Engineering

Lesson 5.1 Introduction to Testing and TDD

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand

Khoury College of Computer Sciences

© 2022 Released under the CC BY-SA license

Learning Objectives for this Lesson

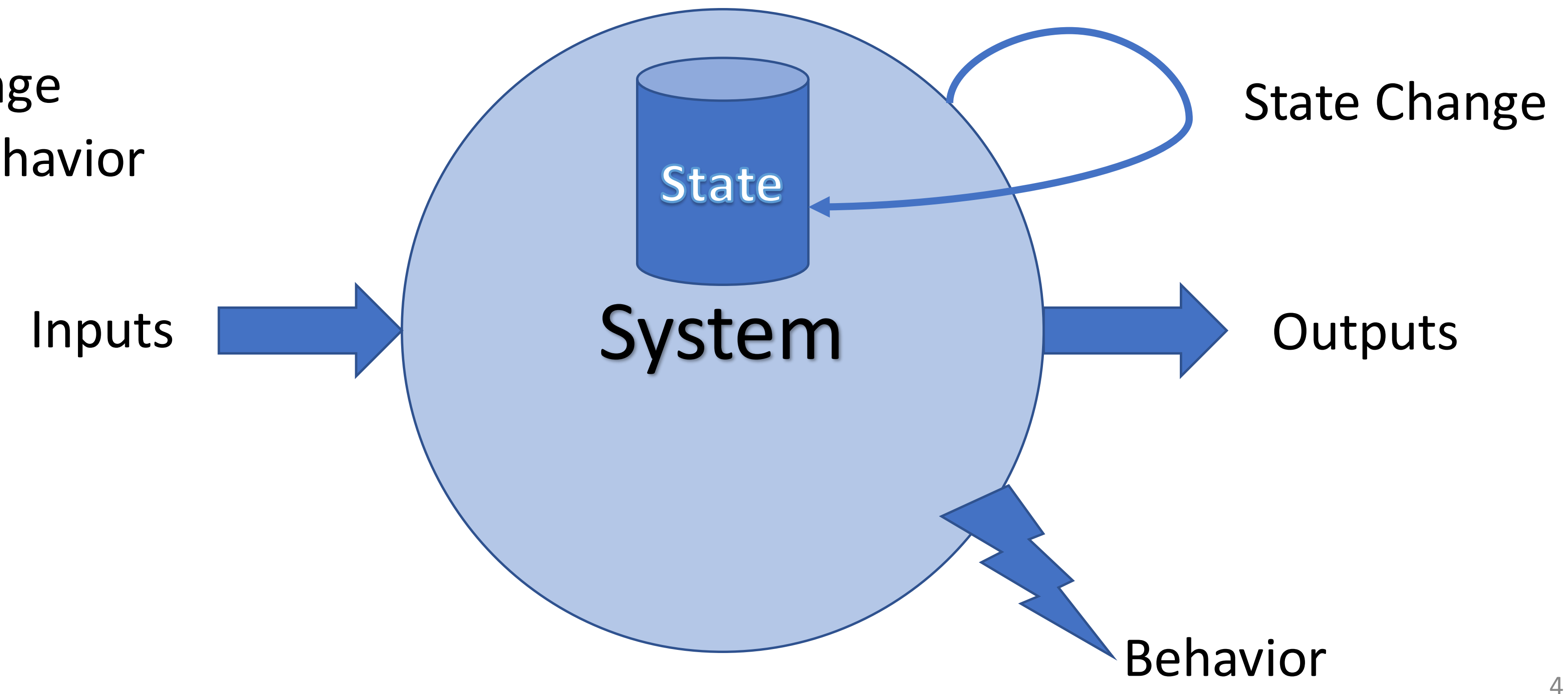
- By the end of this lesson, you should be able to:
  - Describe the elements of a test and how they are used;
  - State Dijkstra’s law and its relevance;
  - Describe how tests are classified by purpose, size/scope, manner;
  - Explain why test automation is important.

Working Definition

- *Software Testing* is the process of checking if software meets **certain concrete** requirements
  - "certain" – a finite set
  - "concrete" – particular, not symbolic
- Testing is carried out by execution of the software.
- Next: definitions "SUT" and "Test"

SUT = System Under Test

- The “System Under Test” consists of its
  - Inputs
  - State
  - Outputs
  - State Change
  - (Other) Behavior



Running a Test

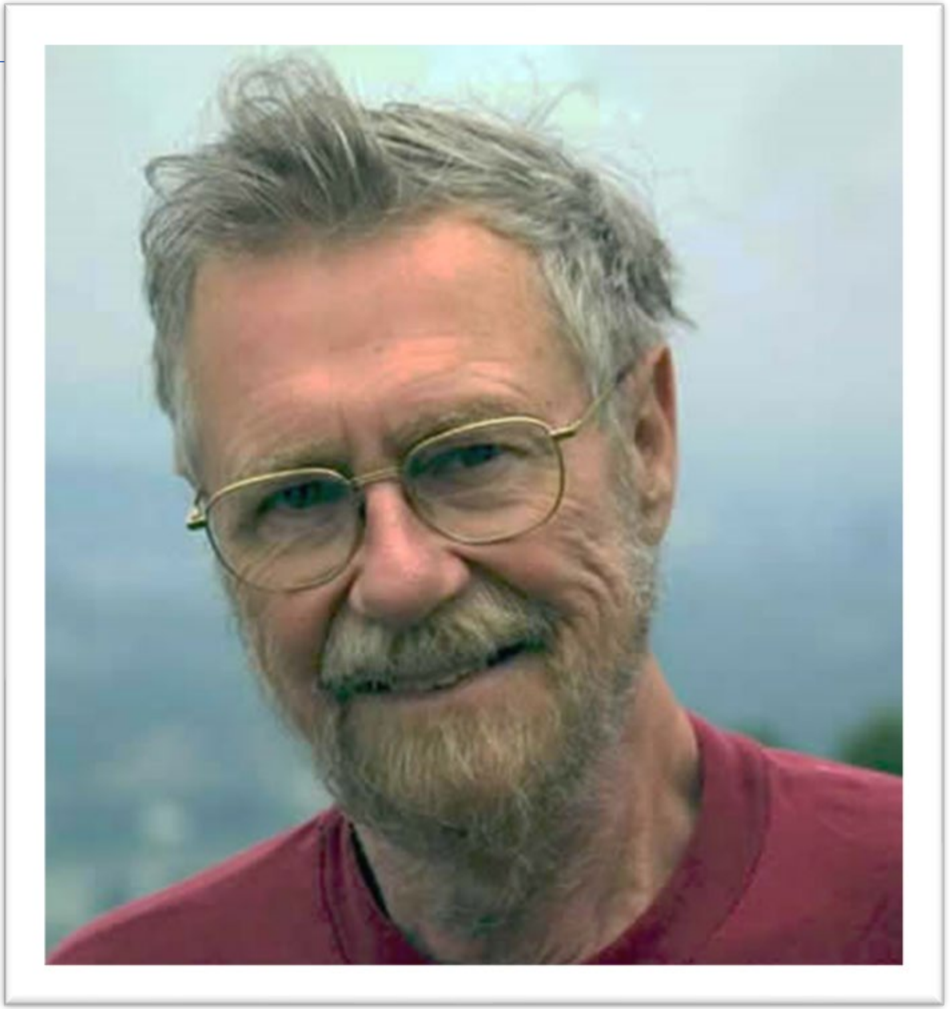
- Construct the situation:
  - Set up SUT to get the state ready
  - [Optional: Prepare collaborators]
- Apply the operation inputs.
- Check the outputs, verify the state change, handle the behavior
  - Handle exceptions,
  - Time-Out to handle nontermination,
  - Post-check with collaborators.

Dijkstra’s Law

“Program testing can be used to show the presence of bugs, but never to show their absence!”
– Edsger Dijkstra

- The state space of a SUT is (usually) infinite, but testing can only execute a finite number of tests.
- Even if the state space is finite, it may still be too large to make exhaustive testing feasible.

And this ignores the fallibility of tests.
What if the tests are in error?



Classifying Tests

- We can classify tests according to several cross-cutting dimensions:
  - Purpose: Why are we testing?
  - Scope: What sort of thing is the SUT?
  - Size: What resources does testing need?
  - Manner: How is testing performed?

Test Purpose

- Test Driven Development
- Regression Test
  - Prevent bugs from (re-)entering during maintenance.
- Acceptance Test
  - Customer-level requirement testing
  - Validation: Are we building the right system ?

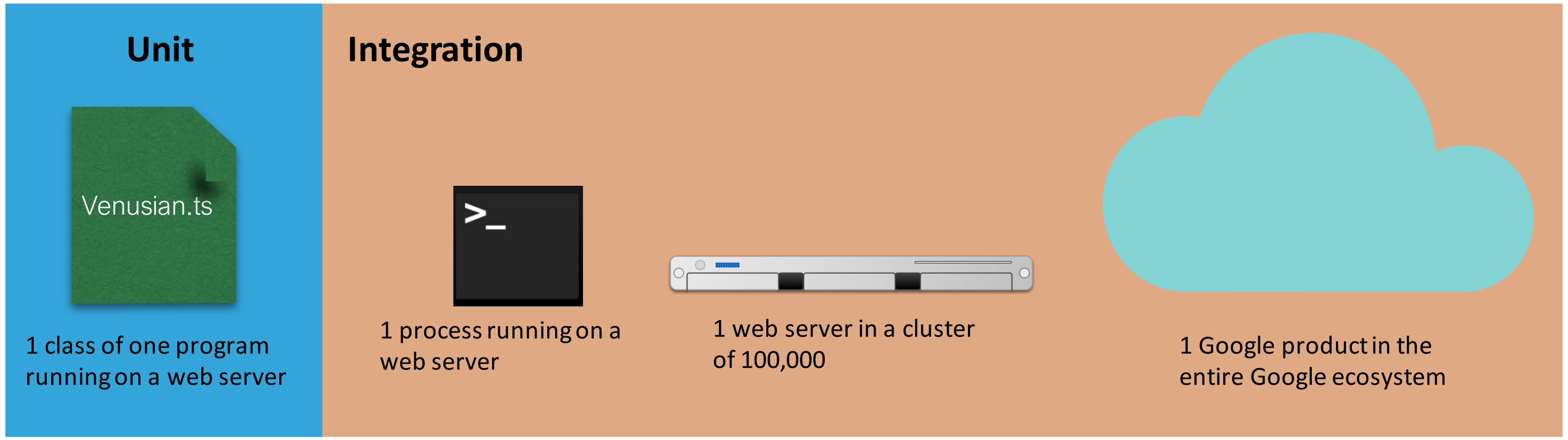
What could a test tell you?

- Function returns the exact “right” answer
- Function returns an acceptable answer
  - Returns the same value as last time
- Function returns without crashing
- Function crashes (as expected)

- Same things for effects, not just returns
  - How it affects the environment

Test Scope

- *Unit* tests: SUT = a single method/class/object
- *Integration* tests: SUT = combinations of units, a subsystem
- *System* tests: SUT = whole system being developed


Unit
Integration
Venusian.ts
>_
1 class of one program running on a web server
1 process running on a web server
1 web server in a cluster of 100,000
1 Google product in the entire Google ecosystem

Test Size follows Scope

We begin by 'testing-in-the-small' and move toward 'testing-in-the-large'

- Small: run on a single process, no I/O
  - Fast to run; can be run automatically and frequently
- Medium: run on a single machine, no network I/O (only localhost); "hermetic"
  - May be slower; delayed to overnight runs
- Large/Enormous tests: run on a network.
  - May have serious $$$ cost in network services or personnel.
  - Much more likely to be "flaky," failures are more difficult to debug

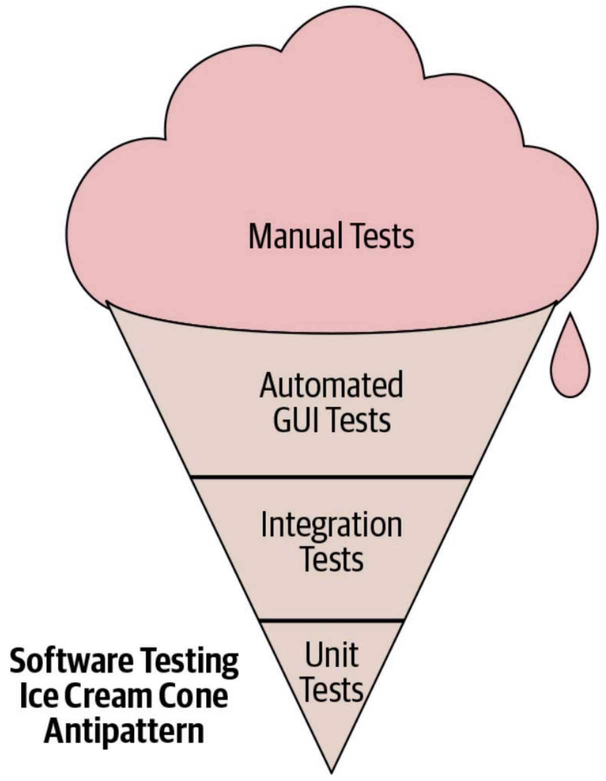
See SoftEng @ Google Chapter 11

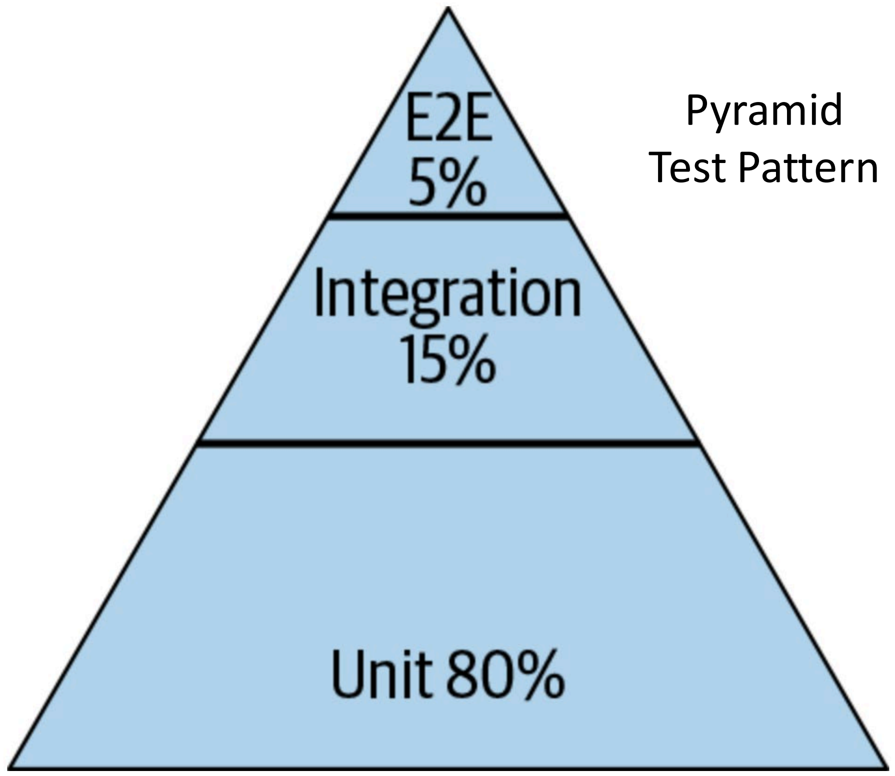
- https://learning.oreilly.com/library/view/software-engineering-at/9781492082781/ch11.html#testing_overview

Manner of Testing

- Black-box vs White-box testing
  - If you have access to source code
- Automated tests can be run without supervision
  - Suitable for frequent automated runs
- Manual tests require a human to run and evaluate
  - A human may be needed to check UI elements
  - Tests may be ill-defined and nondeterministic
    - e.g., trying to “break” software
- Customer-facing tests require an intermediary to evaluate as well as the customer to use the software.

Testing Distribution (How much of each kind of testing we should do?)





Pyramid
Test Pattern

From SoftEng @ Google Chapter 11

- https://learning.oreilly.com/library/view/software-engineering-at/9781492082781/ch11.html#testing_overview

Caveats & Qualifications

- Typically, a new feature will require multiple tests
- The "fix" should not just be the minimum to pass some given set of test(s)
  - The programmer should keep in mind the spec/requirements.
  - But the fix should be the simplest possible that addresses the issue.
- Tests are run frequently and thus must be fast and deterministic.
- Occasionally, the tests may need to be fixed as well.

Review

- Now that you've studied this lesson, you should be able to:
  - Describe the elements of a test and how they are used;
  - State Dijkstra’s law and its relevance;
  - Classify tests by purpose, scope and size;
  - Explain why test automation is important.