CS 4350: Fundamentals of Software Engineering
CS 5500: Foundations of Software Engineering

Lesson 5.4 Testing Systems

Jon Bell, John Boyland, Mitch Wand

Khoury College of Computer Sciences

Outline of this lesson

1. How can we test complex systems?
2. What are some ways to substitute out parts of the system?

Learning Objectives for this Lesson

- By the end of this lesson, you should be able to:
  - Contrast “mocks” and “spies” in testing;
  - Describe the limitations of automated testing;
  - Give some useful examples of nondeterministic testing.

Large Systems are Hard to Test

- Database component
  - Contents may need to reflect/simulate real-world;
  - Data may be expensive/proprietary/confidential.
- Network connections
  - ”Real” connections may be slow/flaky/disrupted;
  - Resources may have changed since test was written.
- Environment
  - Interactions with OS, locale or other software.
- Human actors
  - Ultimately unpredictable.

Testing(!) framework “jest” didn’t install because of Turkish locale (Piazza @108 sp21)

Two Ways to Handle Difficulties

Pay the cost, do the test

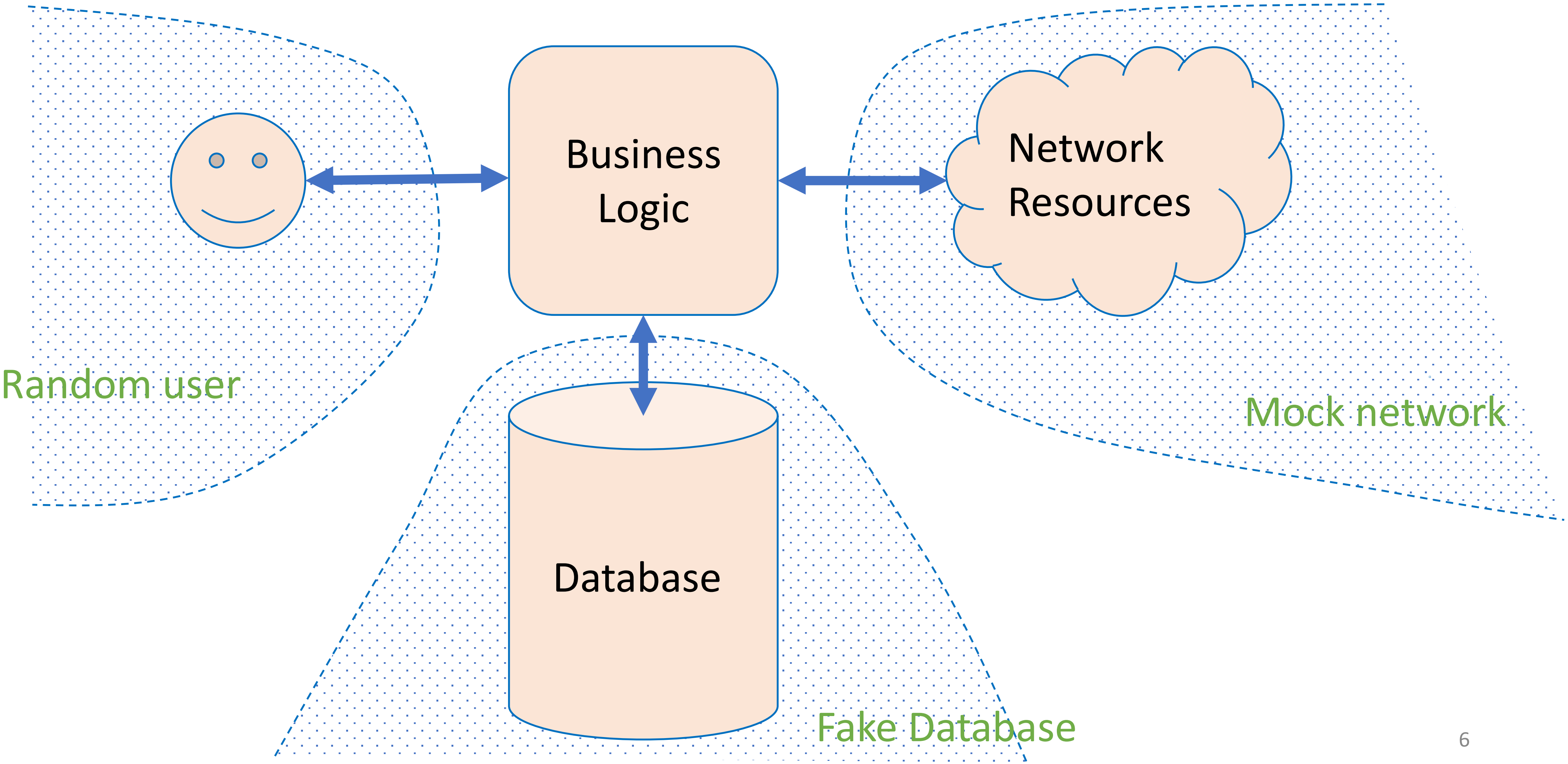
- A large test can reveal problems that smaller tests can't.
- Choose particular times (rare!) to do particular large tests.
- An "enormous" test at Google simulated an earthquake in Mountain View, CA.
- See Chapter 14 of SE@Google

Automate with tools:

- Use "Test Doubles"
  1. Stubs
  2. Mocks / Spies
  3. Fakes
- Random testing
  - "Fuzzing"
  - Against a reference implementation.

Rest of Lesson

Test Double Example



Test Stub

- Supply an object with the same interface:
  - Same methods;
  - Default result values.
- The stub gets the test to run:
  - If the client blindly uses the stub, it can proceed;
  - If the client expects something from the object, the test will likely fail.
- Need two more things:
  1. Remember how the stub was used;
  2. Tell the stub what to do when it is called.

Test Spies

- A test spy remembers how the object was called
  - Then the test harness can check what happened;
  - For example: a particular method should be called
    1. First with parameters “foo” and 42;
    2. Then with parameters “quux” and -88.
- A spy can be useful on the “real” object:
  - What was sent on the network?
  - How many times a problem was logged?
  - What was inserted in the database?
- But most often used with a “mock.”

Test Mocks

- A test mock has scripted results:
  - If such-and-such a method is called
    - return some particular value.
- A complex mock can have many scripts:
  - Multiple methods;
  - Different results for subsequent calls.
- Useful mocking assumes we know how mocked object will be used.
- If a “mock” has real logic, it becomes a “fake”.

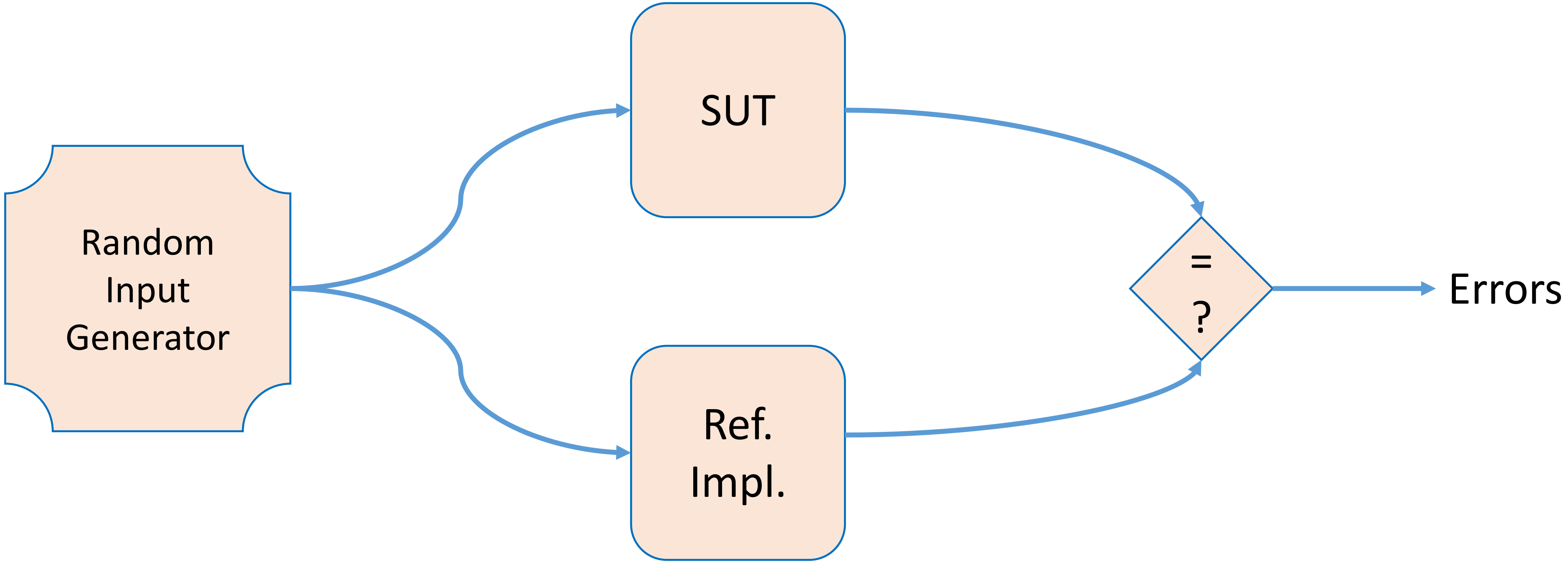
Test Fakes

- A *fake* has an implementation of the object being replaced
  - A *low-fidelity* fake implements things partially
    - Enough to work for the test.
  - A *high-fidelity* fake implements most aspects:
    - Usually all functional aspects;
    - Usually not as efficiently or as scalable.
- The purpose of the fake is to avoid processes/network/cost:
  - So the test can be cheap and deterministic.

Random Input

- To replace a user, we can program a “bot”
  - Randomly use mouse, press buttons;
  - Arbitrary text;
  - Fast or slow.
- Smarter (“Fuzzing”)
  - Capture real actions;
  - Then make targeted mutations.
  - (This applies also to programs taking text input.)
- Expected result can only be imprecise:
  - E.g., “not crash” or “not leak secrets”.

Related: Random Testing



Weaknesses of Test Doubles

- The Mock/Fake may not behave correctly
  - The test harness may assume wrong behavior;
  - Particularly an issue if original object changes
    - Mocks have to be maintained as well!
  - Solution: Test the mock/fake against a higher fidelity fake, or against the real thing.
- The SUT may use a different algorithm:
  - The Spies expect a particular usage of double;
  - The test is “brittle” because it depends on internal behavior of SUT;

Review: Learning Objectives for this Lesson

- You should now be able to:
    - Contrast “mocks” and “spies” in testing;
    - Describe limitations of automated testing;
    - Give some useful examples of nondeterministic testing.

Looking forward…

- In our next lesson, we'll discuss designing for the user experience.