COR 4

Strategies for isolating units

1. Get rid of layer dependencies
    - Call each layer in sequence, passing one's output as the next one's input
    - Prevents any layer from tightly coupling to another
    - Top layer that chains together other layers has no logic and so needs no unit tests
    - *Very hard or impossible in many languages*
    - *Easier in functional languages and ones with evented I/O*
2. Inject your dependencies
    - Make each layer conform to an interface
    - Pass instances of lower layers as parameters to higher ones
    - When testing, can pass *mocks* instead
    - *Hard to do in non-object-oriented languages*

Maxim: avoid round trips

```
UTEST(DatabaseTests, RecordSerialization) {
  Database db;
  db.open("/tmp/mydb");
  Person p;
  p.setAge(17);
  p.setHeight(160);
  p.setName("Bjorn");
  db.addRecord(&p);
  db.flush();
  db.close();

  std::string result = slurpFile("/tmp/mydb");
  EXPECT_STREQ(result.c_str(), "Bjorn;17;160\n");
}
```

demo time!

The Database interface

```
class Database {
public:
  ~Database();

  bool open(const char *filename);
  bool close();
  void addRecord(const Person *p);
  bool flush();

private:
  std::string serializeRecord(const Person* p);
  bool writeRecord(const Person *p);

  FILE *fp_{nullptr};
  std::vector<Person> records_;
};
```

A smaller serialization test

```
UTEST(PersonTests, SerializeSeparatesFieldsWithSemicolon) {
  Person p;
  p.setAge(17);
  p.setHeight(160);
  p.setName("Bjorn");
  std::string result = p.serialize();
  EXPECT_STREQ(result.c_str(), "Bjorn;17;160\n");
}
```

A dependency injection demo

```
class Filesystem {
public:
  Handle *open(const char *filename, const char *mode);
  void truncate(Handle *handle);
  ssize_t read(Handle *handle, unsigned char *buf, size_t nbytes);
  ssize_t write(Handle *handle, Blob b);
  void close();
};
```

demo time!

Self-contained units

Some units, like isEven() are self-contained:

- Don't rely on any external factors
- Results depend only on input
- Similar idea to *pure functions* in functional programming

Most units aren't, though:

- Can produce different results based on external factors
- Layers that perform network, or device I/O may behave differently each call
- Layers that use global state (like the filesystem or global variables) may do the same

Such units aren't self-contained, nor are units that call them.

Maxim: avoid state

- I/O side effect
  - `void computeAndPrintResult();`
- Randomness
  - `if (probabilisticEvent()) doSomething();`
- Filesystem
  - `fwrite(fp, "hello\n");`
- Network
  - `connect()` / `write()` / `close()`
- Database
  - `INSERT` / `UPDATE` / `SELECT`

Writing integration tests

- No need for isolation between your own units
- Need to stop somewhere, though!
  - You can't test the entire world
  - Don't want someone else's web service going down to break your tests
- Can mock things like the filesystem and network
- The scope of an integration test is very project-dependent

Simulating things outside your program

In the same way you can mock out pieces of your code for unit tests, many tools exist to mock pieces of your environment for integration tests.

- Web frontend apps
  - Firefox, Chrome, and others can run *headless* (without a user interface).
  - In this mode, mouse movements, clicks, and more can be triggered automatically.
  - Scripts can use the WebDriver API to perform these actions.
- Mobile apps
  - Android's Espresso and iOS's XCTest frameworks allow simulating user interaction.
- Network services
  - Since network boundaries are well-defined, projects will often write a custom server that always returns fixed responses for the purposes of testing a client, and vice versa.
- System software
  - Can simulate specific syscall or native library behavior or failures using library injection; e.g. LD_PRELOAD on Linux.

Testing at scale

When you change your software, do you run the tests of everybody who uses your software?

- Case study: Rust and other PL infra
  - Rust compiles many open source crates with every version release as regression testing
- Case study: big companies like Google/Meta
  - Run all of your tests and tests of dependent projects

What do you do if you find breaking behavior surfaced by a large integration test written by your client?

Doxygen

- Generate human-readable documentation from inline comments
- Supports multiple languages (C, C++, Python, Java, and more!)
- Lets people browse inline API documentation from the web (or man pages, or…)
- You will be reading a Doxygen-generated website for HW8