Designing and Testing Software

BY EVAN OLDS

CPT S 422

Priorities

1. Above all else – solid knowledge of how code works
   - Know all the features of language you're using
   - Know what's going on beneath your code – things that are done automatically for you
     - Standard library stuff
     - Compiler-hidden stuff like destructor calls and deep vs. shallow copies
2. Design the software itself as something that's testable
   - If it's not flexible enough, you may not even be able to test the necessary components
   - Much more on this later

Priorities

3. Understanding of testing
   - concepts and terminology
   - strategies/methods
   - tools
   - frameworks
   - (much more…)

Flexibility

- Before investigating a bunch of test strategies, methods, tools, and frameworks, first think of the design of the actual software
- Break it down in to logical components
- After writing some code check - do the “conceptual” components match the actual implemented components?
- Examining scenarios will help paint a picture of this (done in class)

Scenario 1 – File Loading and Saving

- This was covered previous
  - Write the software to use streams
  - Save to a stream
  - Load from a stream
- Can write stream classes specifically designed for testing purposes
- As one example: suppose you want to test how well your software loads from a stream that has slow reading functionality
  - Happens in practice if a system is really bogged down, but before solution details, discuss in class why a single system can be running quickly at one moment and slowly the next

Scenario 1 – File Loading and Saving

- The problem: suppose you want to test how well your software loads from a stream that has slow reading functionality
  - Hard to design a test to *intentionally* bog down the system and reproduce those "stressed system" conditions
    - As an aside, "stress testing" is a real thing and is very common for production software
  - Wouldn't work anyway if you start running your tests on more powerful hardware
  - Turns out we don't need to actually stress the system at all to see how it might behave when there is system slowdown
  - Why? How do we test for these types of scenarios?

Scenario 1 – File Loading and Saving

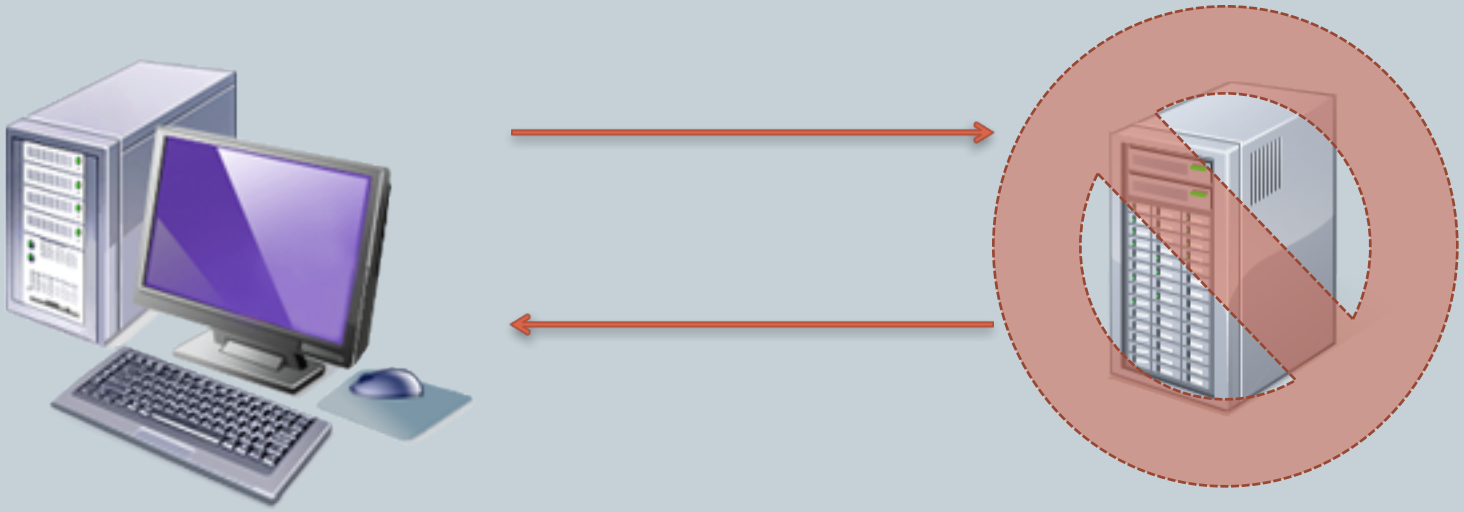
- The problem: suppose you want to test how well your software loads from a stream that has slow reading functionality
- The answer: just write a "SlowStream" class that intentionally takes a long time in a Read function
- Have it sleep for 5 – 10 seconds before filling the buffer and returning
  - It can be that simple
  - As far as the loading code in the product is concerned, this stream may behave similarly to a FileStream on a system with high disk usage
- In actual product code build: FileStream passed to Load function
- In test suite code: app engine loaded from DLL, SlowStream passed to Load function, functionality analyzed

Scenario 2 – Database Source… for which there is no database

- This scenario is a bit more involved (or is it?)
- More effectively illustrates the importance of the design of the software
- Assume:
  - Building app that gets data to display in the UI from a database
  - Database will eventually consist of a collection of records about students at a university
  - Interface between app and DB is known, as is desired XML data format

Scenario 2 – Database Source… for which there is no database

- Assume:
  - Database team is behind schedule and you have to start developing (and testing!) the UI before they get the database online



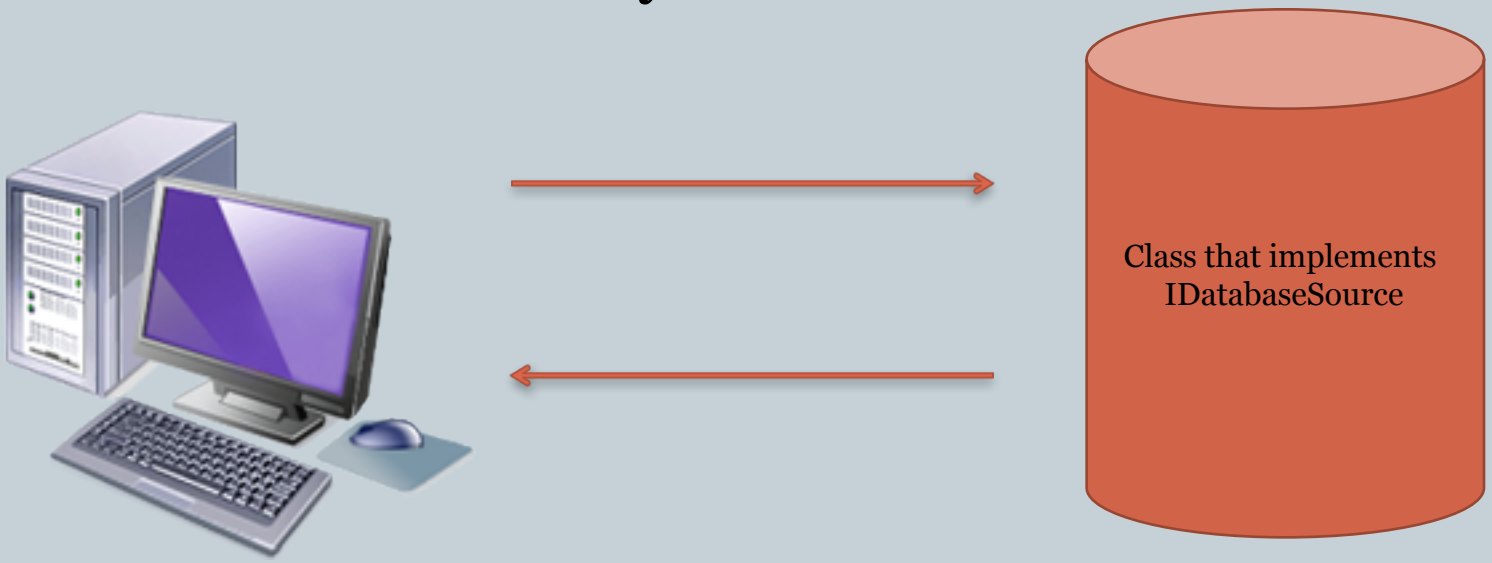
  - What can we do (other than wait)?

Scenario 2 – Database Source… for which there is no database

- Answer:
  - Code that loads from database is designed to load from IDatabaseSource (or abstract base class)
  - Eventually there will be a class that implements IDatabaseSource and connects to the actual database once it exists
  - In the meantime write a SampleDatabase class that implements IDatabaseSource and delivers sample data from
    - test files
    - and/or
    - procedurally generated data
- In other words “simulate” the database via utilizing object-oriented abstractions

Scenario 2 – Database Source… for which there is no database

- If the right abstractions are in place in the code, a testing component can be plugged in where the database will eventually be



Scenario 3: Test Invalid HTTP Request

- Say you have code to accept a connection and read (what might be) an HTTP request

```
var client = AcceptClient()
NetworkStream ns = client.GetStream
// Can read from ns here…
```

- If we initiate the connection from the browser, it will send a valid HTTP request
- How do we test code that's designed to handle invalid HTTP requests? (discuss in class)

Scenario 4: Testing a Spreadsheet Application

- Assume we're building a spreadsheet application at a corporation in the industry. How do you test it?
  - Want to make sure we test it in an automated fashion
  - In other words, how do we test it with code?

Scenario 4: Testing a Spreadsheet Application

- A variety of places you could start, one approach isn't necessarily better than others
- Assume you've already partitioned your project into pieces and have assigned responsibility for those pieces to different members in the group
  - One option is to have every person that's responsible for designing a particular piece/feature also responsibility for generating tests for that piece/feature
  - Another is to intentionally have different developers testing features that they didn't build.
  - Could also have X% of the team developing components exclusively and 100-X% developing just the test framework

Scenario 4: Testing a Spreadsheet Application

- What if you don't have your project partitioned into pieces and different roles assigned to different developers?

Scenario 4: Testing a Spreadsheet Application

- What if you don't have your project partitioned into pieces and different roles assigned to different developers?
- Then do it
- Will be hard to develop tests for a system if you can't break that system into small pieces
- If it's difficult to break up the application into pieces then this should be a warning flag
  - You're not choosing a good design and likely will have high coupling and code that's hard to manage

Scenario 4: Testing a Spreadsheet Application

- But back to the question: how do we test the spreadsheet application?
- Let's start by identifying what should be individual components
  - Should be decoupled from each other to some extent
  - Individually testable (let's start with unit tests)
  - We'll consider testing interaction between components in a bit
  - For now let's just identify pieces

Scenario 4: Testing a Spreadsheet Application

- If it's hard to identify components then maybe switch the terminology and ask:
- What are the features of this application?
- Features often map fairly directly to components
- So what are the features of a spreadsheet application?

Scenario 4: Testing a Spreadsheet Application

- So what are the features of a spreadsheet application?
- What unit tests should we write for the features that we've identified
- (in class development of unit tests)

Scenario 5: Image Processing / Object Detection

- You're writing code to control a robot, drone, or some other entity that has a camera
  - Identifies objects in images captured by the camera
  - Makes decisions on what to do based on object detection
- To simplify the scenario, you could just suppose that you have a webcam attached to a laptop and it needs to be able to identify objects
  - Such image processing algorithms (object identification) are not easy
  - We're not going to discuss the implementation details of such algorithms, but needless to say you'll want test code for such algorithms

The Question

- How do you test this?
- A better question is how do you test it with an automated testing framework?
- Automated test frameworks can be run on a regular basis
  - Probably at least weekly, if not every single night after employees go home
  - Errors that are found are flagged, and the next morning when employees come to work they see the list of tests that failed and can investigate if need be, or move on with their work if all tests pass

The Question

- But to get this automated testing of image processing / object detection algorithms in place, we need to consider a few things:
  1. How are we going to set up the test environment if we need on object with a camera mounted to it to be moving around or perhaps the camera can be fixed but objects in front of it have to move around?
  2. How do we verify (remember: in an automated way) that our algorithm is properly detecting the right objects in the right way?

The Design

- First off, throw out the idea that you need to have a camera capturing images of objects to test your object detection algorithm. This is not needed to get a lot of automated testing in place.
- As with several things we've already encountered in this class, flexible design in the software will help facilitate automated testing.
  - Abstraction
  - Can swap out actual product components with testing components easily

The Design

Have an abstract base class (or interface if using C#/Java) representing an image capture device

```
interface IImageCapture
{
        public abstract Image CaptureImage();
}
```

The Design

Have a class that inherits for actually capturing images from a hardware device (the camera actually attached to the robot)

```
class CameraImageCapture : IImageCapture
{
    public Image CaptureImage()
    {
        // Code to interface with the camera device and
        // get the image. Then “package it” and return it.
    }
}
```

The Design

Have another class (for testing) that inherits from IImageCapture and delivers images (we'll discuss how shortly) that are used for automated testing

```
class TestImageCapture : IImageCapture
{
    public Image CaptureImage()
    {
        // Examples on next slides
    }
}
```

The Design

Although this is not likely to be doable with most projects that have a significant level of complexity in their image analysis scenario, for very simple cases you may be able to have a test class that procedurally generates the images.

```
class AutoGeneratedTestImgs : IImageCapture
{
   public Image CaptureImage()
   {
      // Render an image in memory
   }
   public ObjectLocationData GetLocationData()
   {
      // Returns object location data corresponding to the objects in the image
      // that was last returned by Capture Image
   }
}
// (demo)
```

The Design

- More complex scenarios can still use a similar design: abstraction on capture source and component to plug in for testing
- If you need analysis of “real” images then use photos similar to what will be analyzed by the application
- There will be a class that implements IImageCapture that delivers images (frames) from a picture files on disk
  - Will also have information about what the “correct” assessment of each image is
  - If the location of an object is needed, then the class should also load that information. The user may have to determine this manually and enter it as part of the test.
- Demo will elaborate on this
- (demo)

The Design

- Can use still images for tests cases. Not uncommon to have data files associated with test cases.
- Can extend this by building tests around recorded video where the expected outputs are entered by the user.
- There will be a class that implements IImageCapture that delivers images (frames) from a video file
- A bit more involved to define correct outputs for the test, but an app could be built to let the user click on objects in various frames from the video and save the (time, location) pairs to a data file. Unit tests load these, interpolate appropriately, and use values for verification.

The Point

- Even image/video processing scenarios where the code has to analyze something in real-time from a camera, you can automate the tests
- Get images or video for the automated tests that you can label beforehand with the relevant properties
  - Could be information about where certain objects are
  - Could be information about how fast objects are moving
  - Could be information about whether or not the robot is about to hit something
  - And much more…
- Layer of abstraction through IImageCapture
  - Designs similar to this will make this doable

Scenario 6: Automated Testing of Mouse Input

- A large percentage of modern applications utilize mouse (or touch) input in some way
- This is another one of those cases where the approach to automated testing may not be immediately clear
  - Don't want to require a human to do testing
  - Don't want anything to require physical mouse movement in the tests, but want to be testing mouse input scenarios nonetheless
- Let's take a specific type of application…

Scenario 6: Automated Testing of Mouse Input

- Suppose our application is a visual drag-and-drop, point-and-click designer of some sort
  - Could be creating process flow diagrams, automata, simple state machines
  - Could be an educational app where you have to drag objects into certain boxes/categories on screen
- Could be a lot of things, but let's say it's primarily mouse driven
- Can you, based on some reoccurring theme in this course, guess where we should start with respect to the test design on this?

Scenario 6: Automated Testing of Mouse Input

- The application itself should have some abstract base class for the input

```
abstract class Input
{
  void OnClick(ClickArgs a);
  void OnMove(MoveArgs m);
}
```

- Even without considering testing, this is often something you want in place

Scenario 6: Automated Testing of Mouse Input

- Input “modes” can change frequently based on the task that the user is doing. These could all be different input modes:
  - Dragging an object from a toolbox into the workspace
  - Clicking on an existing object in the workspace and dragging it to a new location
  - Drawing some sort of connection between objects
- Obviously the input logic in each of these cases in slightly different
- Each above item could have its own class inheriting from the Input base class

Scenario 6: Automated Testing of Mouse Input

- Main input processing window has a reference to an Input object: m_input
- Creates a new instance as needed based on the user starting a new action
  - User clicks down on existing object in the workspace → m_input = new DraggingInputState(objectThatWasClicked)
  - User clicks down on an item in the toolbox → m_input = new CreatingNewObjectState(toolboxItemType)
- Separating different types of input logic into different classes

Scenario 6: Automated Testing of Mouse Input

```
abstract class Input
{
  void OnClick(ClickArgs a);
  void OnMove(MoveArgs m);
}
```

- Separating different types of input logic into different classes
- If you've ever made a drag-and-drop editing application, you know that managing input states gets messy if you try to do it all in one place
- Also, we can generate unit tests that test these objects individually. They "simulate" mouse events simply by calling the click or move events. Can check to ensure that the right objects were created/altered within the workspace.

Scenario 7: Poorly Design Apps + No Control

- For the majority of final projects in this class, you're building the app for another class and thus have control over the design.
  - I expect to see good design that allows for a good test framework
- For those of you that are testing an app that you can't modify, there's a potential issue
- What if the app doesn't have a good interface for you to inject your test components?
- Should still be able to do unit testing on certain pieces

Scenario 7: Poorly Design Apps + No Control

- What if the app doesn't have a good interface for you to inject your test components?
- Should still be able to do unit testing on certain pieces
- Also, if nothing else there should be some way to do automated functional/black-box tests

Scenario 7: Poorly Design Apps + No Control

- Automated functional tests:
  - If app is command line, spawn it as a process and verify its output
    - Could redirect stdout if you need to verify that
    - Could check for correct data files if app is expected to produce them
  - If app is web-based can test web services by generating web requests from a standalone testing application
    - Verify that a request that's supposed to return specific XML data actually returns in under a variety of conditions
    - Verify that requests made without proper authentication (if secure service) don't return sensitive data

Scenario 7: Poorly Design Apps + No Control

- There is almost always a way to test from the outside of the app
- It's not just when you don't have permissions to modify the project code. I expect automated functional tests to some extent on most projects.
- But in the case where the app is really not extensible for testing purposes, you'll have to figure out what do do
  - Think acceptance testing: you ask "what is the app supposed to produce for the user" and you provide that simulated user input and see if you get the expected result

Scenario 7: Poorly Design Apps + No Control

- One last word on poor design for scenarios that may occur outside of this class
- If an organization recognizes that an app is poorly designed and its hindering project progress, they should invest in fixing it instead of just trying to build more and more on top of it
- This is an opinion, but working with convoluted code wastes a ton of developer time. Even if it means you aren't going to make progress on new features for a while, could be worth it to take the time to fix the foundation.
- Much faster progress when building the new features on top of a solid foundation.