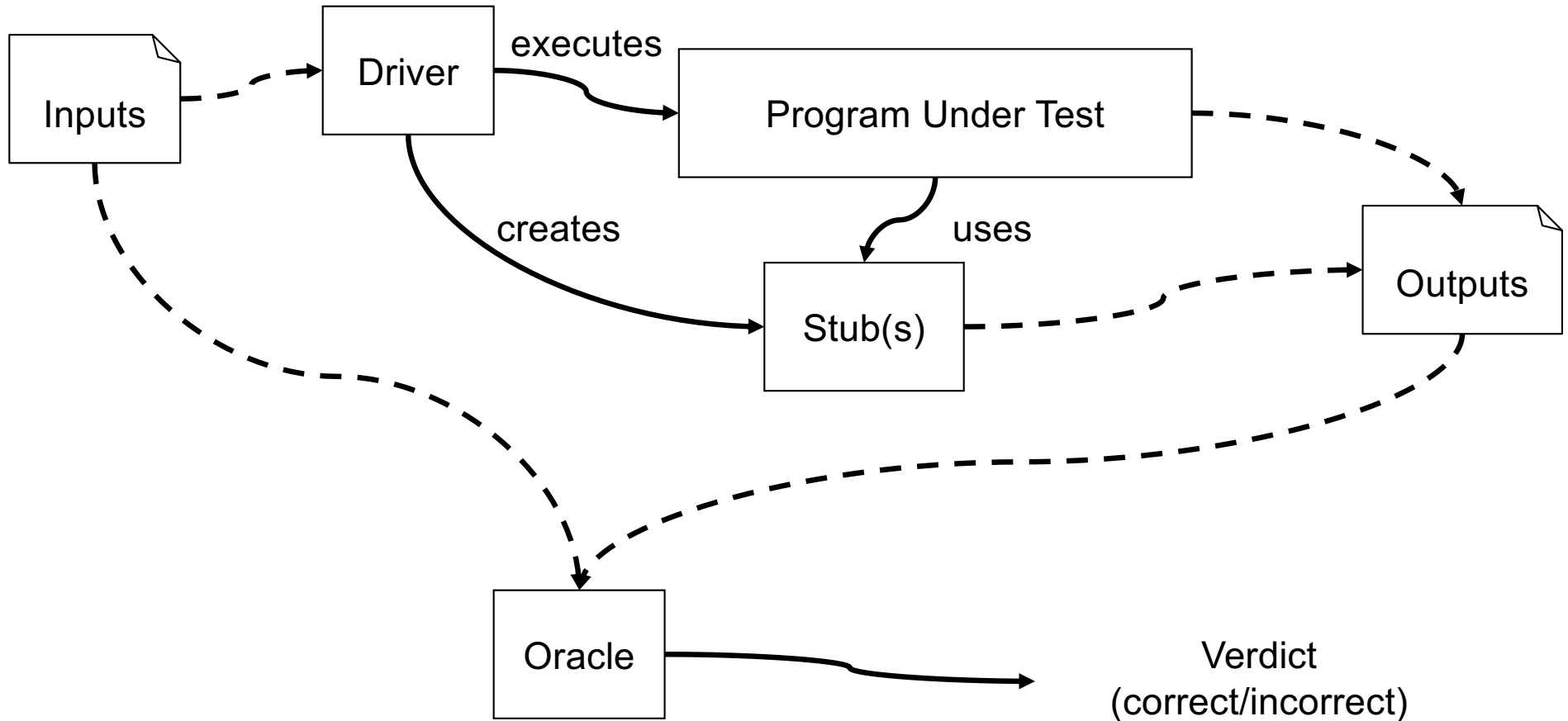

SYSC 4101 / 5105

Drivers, Stubs, and Oracles by Examples (with automation in mind)

Driver / Stub / Oracle



Driver / Stub / Oracle—a simple example

```
import junit.framework.*;

public class TestMyClass extends TestCase {

    public void testOutputSequence() {

        stub = new stubServer( setup parameters );
        client = new MyClass(stub);
        client.call( parameters );
        String res = stub.getSequenceOfCalls();
        assertEquals(res, "blablabla");

    }

    ...

}
```

- We want to check that the right sequence of messages is sent to the stub server.
- We do not need (or have) the actual server.

This is a **Stub**.

The driver performs the test.

This is an **Oracle**.

This is a **Driver**

Establish a link between the class under test and the stub.

Drivers, Stubs, and Oracles by Examples

- Drivers
- Stubs
- Oracles

Drivers

- Assume you have a `diff` operation that computes the difference between two ordered sets (i.e., elements that are in the first set but not in the second)

```
// Java code chunk
```

```
OrdSet s1, s2, s3;
```

```
...
```

```
s3 = s1.diff(s2);      // s3=s1`s2, e.g., {1,5}={1,4,5}`{4}
```

```
System.out.println(s3);
```

- Assume you devised numerous test cases (data).
 - What are possible strategies to write the test driver(s)?

Drivers - First solution

```
public class Driver {  
    public static void main(String argv[]) {  
        // The test case consists in s1={1,4,5} and s2={4}  
        OrdSet s1 = new OrdSet(); OrdSet s2 = new OrdSet();  
        OrdSet s3 = new OrdSet();  
        s1.add(1); s1.add(5); s1.add(4);          // adding elements to sets  
        s2.add(4);  
        s3 = s1.diff(s2);  
        System.out.println(s3);  
    }  
}
```

- **Solution:** writing all the test cases in function main in the driver.
- **Can't selectively run test cases** (functional vs performance testing, regression testing)
- **Can't have different test sets for a class**

Drivers - Second Solution

- One static method per test set (namely TS1, TS2, ...)
- One static method per test case (namely TC1, TC2, ...)
- Test sets can share test cases

```
public class Driver {
    public static void main(String argv[]) {
        TS1(); // Test set 1
    }
    public static void TS1() {
        TC1();
        TC2();
        ...
    }
    public static void TS2() {
        TC5();
        TC2();
        ...
    }
}
```

```
public static void TC1() {
    OrdSet s1 = new OrdSet();
    OrdSet s2 = new OrdSet();
    OrdSet s3 = new OrdSet();
    s1.add(1); s1.add(5);
    s1.add(4);
    s2.add(4);
    s3 = s1.diff(s2);
    System.out.println(s3);
}
public static void TC2() {...}
...
}
```

Discussion

- All the (static) methods that execute test cases have (usually) the same structure
 - Here: creating sets s1 and s2, and calling the diff
- A testing technique can produce hundreds of test cases
- What happens if we want to add test cases (test sets)?
 1. Add new TCxxx (or TSyyy) static methods
 2. Add calls to these methods
 3. Compile the driver
 4. Execute the driver
- Create different tests for different purposes, e.g., successive regression test sets
- Can change one statement in the `main()` to execute different test sets (but need to recompile and re-execute)

Drivers - Third Solution

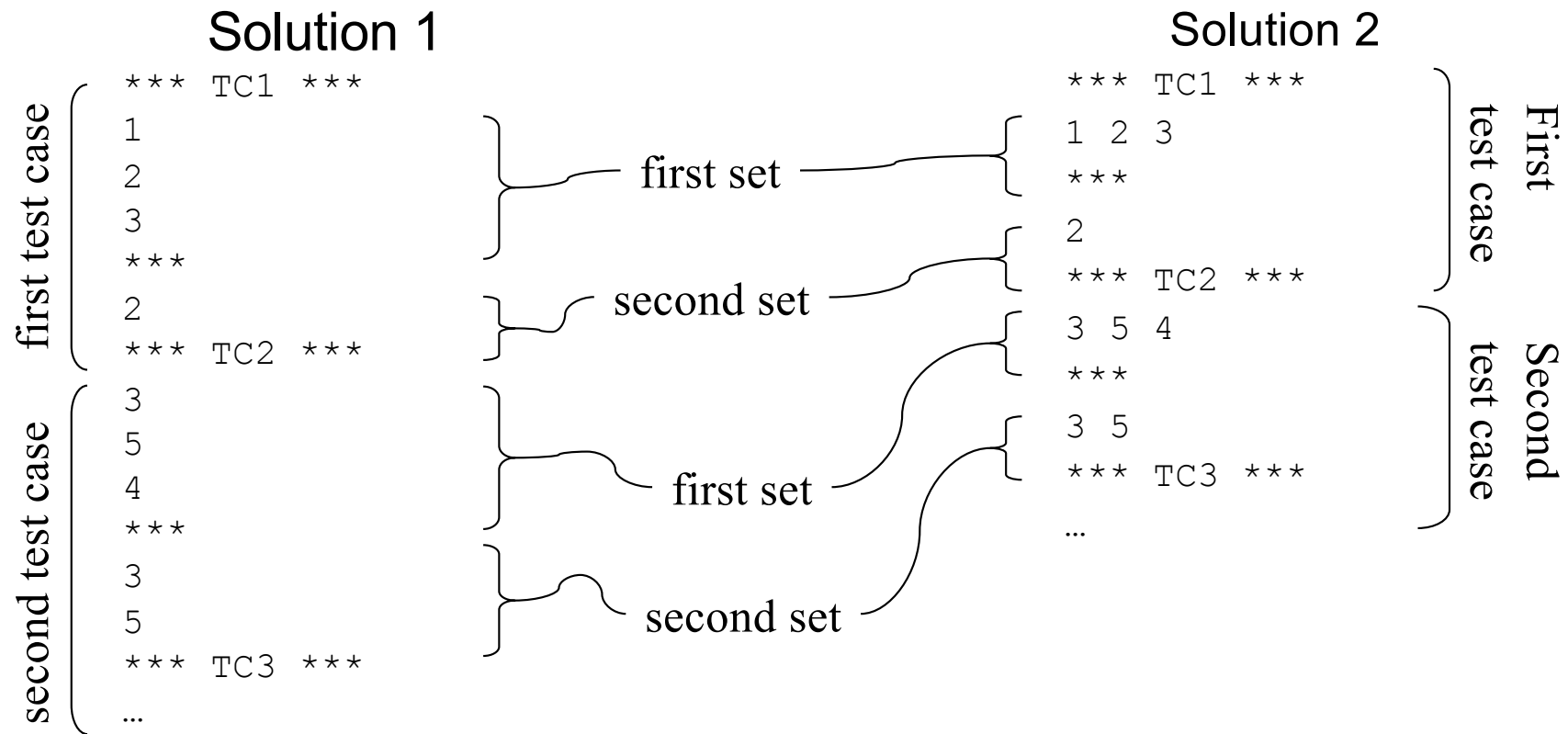
- The main function in the driver reads a text file:
 - 1 test set per text file
 - The driver must read the text file, create the objects according to its contents and execute the diff

```
public class Driver {
public static void main(String argv[]) {
    OrdSet s1, s2, s3
    // open file and read test set name ...
    while (not at the end of file) {
        // 1. read a test case
        // 2. instantiate s1 and s2
        // 3. execute the diff
        // 4. output the test case number and the result
        // the last 3 steps can be in a separate (static) method
    }
}
```

Discussion

- What is the format of the text file (we have to identify the test case data)?
 - Many test cases in the text file
- The driver (the main) needs to:
 - Identify test cases:
 - Beginning and end of a test case in the file
 - Number/name of the test case
 - Know how to read the data in test cases
 - Separation between different information in the test case (e.g., the two sets)
- Different strategies
 - Ad-hoc strategies
 - xml file
 - ...

Drivers - Third Solution



More Complex Drivers

E.g., test cases are different (different kinds of executions/data)
(possible) Solutions

1. Several different drivers (main functions) with different execution flows (i.e., with different file format)
 - Driver (main) one: file format 1
 - Driver (main) two: file format 2
 - Implementation/usage depends on the programming language (difference between Java and C/C++)
2. One driver with several possible inputs (e.g., command line arguments, or in the text file)
 - Driver (main) with input 1 (command line argument): file format 1
 - Driver (main) with input 2 (command line argument): file format 2
 - Implementation language independent (we pass information on the command line)

More Complex Drivers (in Java)

- In Java, we can have different driver classes, each having a main() function.
 - In the same directory we would have files OrdSet.java, Driver1.java, Driver2.java, ...
 - And Driver1.java, Driver2.java, ..., have a main function
- We just choose which main (driver class) we want to execute when executing the Java Virtual Machine
 - java Driver1 [...]
 - java Driver2 [...]
 - ...
- Solution can be used even if we test a program (not a class), that already has a main function
 - Drivers can be considered as additional “entry-points” in the program

More Complex Drivers (in C++)

- Conditional compilation in C/C++
 - Enables the programmer to control the execution of preprocessor directives and the compilation of program code.
 - Using `#define`, `#ifdef`, `#ifndef`, and `#endif` preprocessor directives, that:
 - Determine whether symbolic constants are already defined
 - Determines whether parts of the code are compiled (and thus executed)
- Two solutions:
 - Remove/Add the preprocessor directives
 - Requires modifying source files and compilation
 - Use compiler options
 - Requires compilation only
- Solutions can be used even if we test a program (not a class), that already has a main function
 - Drivers can be considered as additional “entry-points” in the

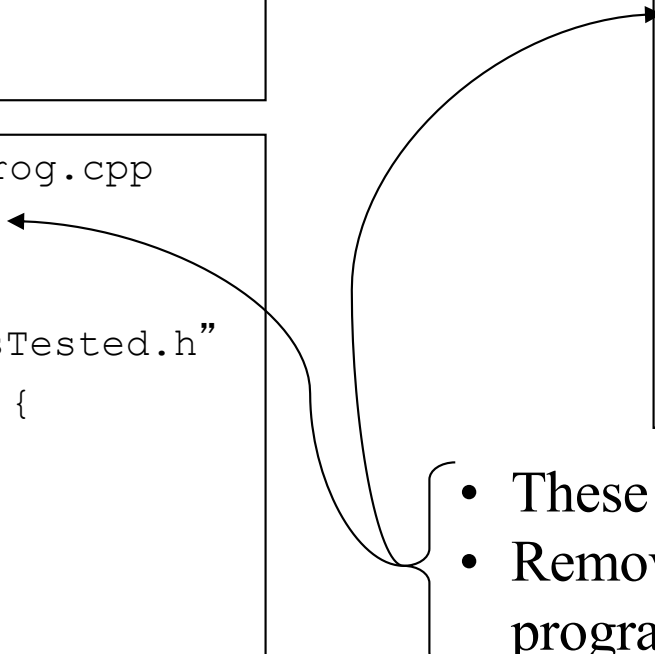
More Complex Drivers (in C++)

```
// File: ClassTester.h
#include "ClassTested.h"
class ClassTester {
public:
    ClassTester();
    void runTestSuite();
private: ...
};
```

```
// File: MainProg.cpp
#define DRIVER
#ifndef DRIVER
#include "ClassTested.h"
class MainProg {
public: ...
private: ...
};
#endif
```

```
// File: ClassTester.cpp
#include "ClassTester.h"
ClassTester::ClassTester() {...}
void ClassTester::runTestSuite() {...}
// code to run and report test cases
```

```
// File: ClassTesterMain.cpp
#define DRIVER
#ifdef DRIVER
#include "ClassTester.h"
int main() {
    ClassTester tester;
    tester.runTestSuite();
}
#endif
```

- 
- These lines make the driver execute
 - Removing them make the main program execute

More Complex Drivers (in C++)

```
// file Main.c
#ifdef DRIVER
#include <stdio.h>
int main() {
    printf("main program\n");
}
#endif
```

```
// file Driver.c
#ifdef DRIVER
int main() {
    printf("driver\n");
}
#endif
```

- Executing the Main of the program
 - i.e., the main in file Main.c

```
cc -c Main.c
cc -c Driver.c
cc Main.o Driver.o -o Exe
```
- Executing the Main of the driver
 - i.e., the main in file Driver.c

(compiler option `-D` defines constant `DRIVER`)

```
cc -DDRIVER -c Main.c
cc -DDRIVER -c Driver.c
cc Main.o Driver.o -o Exe
```

Automated Driver Generation?

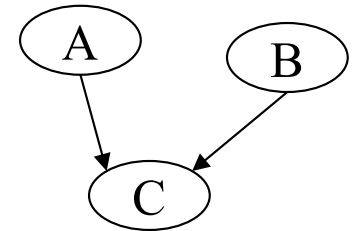
- Technology exist for specific domain, problems
 - Below are only a sample set of technologies
- For GUI testing
 - Record and replay software
 - E.g., Testar
- For protocol testing
 - TTCN-3 (though it is no longer specific to protocol testing)
- For web applications
 - Selenium

Drivers, Stubs, and Oracles by Examples

- Drivers
- Stubs
- Oracles

Stubs

- Need for stubs:
 - Parts of the system that are not yet unit tested or even available (i.e., the code is not ready)
 - E.g., simulation of hardware devices
- Example 1:
 - Modules A and B use services provided by module C.
 - Here, modules can be functions, classes, sub-systems
 - But: A uses only part of the services in C, say C_A
B uses only part of the services in C, say C_B
 - Then: When testing A, we create a stub simulating C_A's behavior
When testing B, we create a stub simulating C_B's behavior
 - We are not simulating (stubbing) the entire C!
- Comment:



“If the stub is realistic in every way,
it is no longer a stub but the actual routine” [Beizer]

Stubs

Example 2:

- An Automated Teller Machine (ATM) has a keyboard (i.e., the hardware device) and a `Keyboard` class.
- During development and testing, class `Keyboard` is not “connected” to any hardware device.
 - Rather, it is used by the driver to “feed” the ATM system with a PIN number, an amount, ...
- Typical test case:
 1. Enter the PIN
 2. Select the account
 3. Enter an amount
 - Each time, the ATM system asks class `Keyboard` for data.
 - Class `Keyboard` must return the correct value according to the test case.

Stubs

- Problem:
 - The driver must set different values during the execution of a test case.
- Solutions for the stub:
 1. The driver must “stop” the ATM in order to tell the stub what is the next value to be used (threads?)
 2. Class `Keyboard` (stub) knows where (e.g., a file) to read the different inputs.
 3. ...

Stubs (responses/behaviors)

- Intent [Binder]:
 - Use a temporary, minimal implementation of a method, or class (a stub) to increase controllability and observability during testing
- Possible responses/behaviors of a stub to the client
 - Return to the client with no action
 - Return a constant value
 - Return values read from a file, a database, or a persistent collection. The values may be constant, selected sequentially, or selected randomly.
 - Return values computed by a simplified or reduced algorithm (simulation of the actual server)
 - Wait or loop for a specified amount of time (to control un-expected delays in a real-time system for instance)
 - Wait for occurrence of some event not under the control of the client
 - Simulate exceptions or abnormal conditions
 - ...

Stubs (responses/behaviors)

- Possible responses/behaviors of a stub to the driver
 - Log or display the values of the input arguments to a file or the test console, or make that information available to the driver upon request
 - Log or display the name of the stub and the type of (client) object to which it was bound
 - Monitor system resources utilization: memory, processes, threads, CPU... Write information to a log or display it, or make it available to the driver upon request.
 - Send a trace message to a file or display, or make the information available to the driver.
 - Generate a stack dump
 - ...

Automated Stub Generation?

- Different infrastructures, tools, framework exist and can be used.
- They only solve part of the problem, providing only a subset of the possible responses/behaviors of a stub (see previous list)
- Example:
 - jMock
 - Mockito

Stubs are also called mock objects

Automated Stub Generation?

A Mockery represents the context, or neighbourhood, of the object(s) under test. The neighbouring objects in that context are mocked out. The test specifies the expected interactions between the object(s) under test and its neighbours and the Mockery checks those expectations while the test is running.

```
import org.jmock.Mockery;
import org.jmock.Expectations;
class PublisherTest extends TestCase {
    Mockery context = new Mockery();
    public void testOneSubscriberReceivesAMessage() {
        // set up stub/mock
        final Subscriber subscriber = context.mock(Subscriber.class);
        Publisher publisher = new Publisher();
        publisher.add(subscriber);
        final String message = "message";
        // expectations
        context.checking(new Expectations() {{ oneOf (subscriber).receive(message); }});
        // execute
        publisher.publish(message);
        // verify
        context.assertIsSatisfied();
    }
}
```

Creates a mock object of type Subscriber.

Establishes a link between the object under test and its stub.

What the stub should see/receive (expected invocation: a call to receive with message as argument)

Execute test case

Oracle

Drivers, Stubs, and Oracles by Examples

- Drivers
- Stubs
- Oracles

The Oracle

- In the previous approaches for the implementation of the driver, the driver reports the observed output.
 - Deciding which test cases failed (a fault has been revealed) is done manually.
 - Comparing the expected output with the observed one, for each test case
 - Two possible definitions of the Oracle
 1. The oracle can decide an expected output corresponds to the actual output *[mostly used definition]*
 2. The oracle can **compute the expected output** (exactly or approximately) given the test inputs and then can **decide an expected output corresponds to the actual output**
- “corresponds” \cong equals, equals with margin for error ...

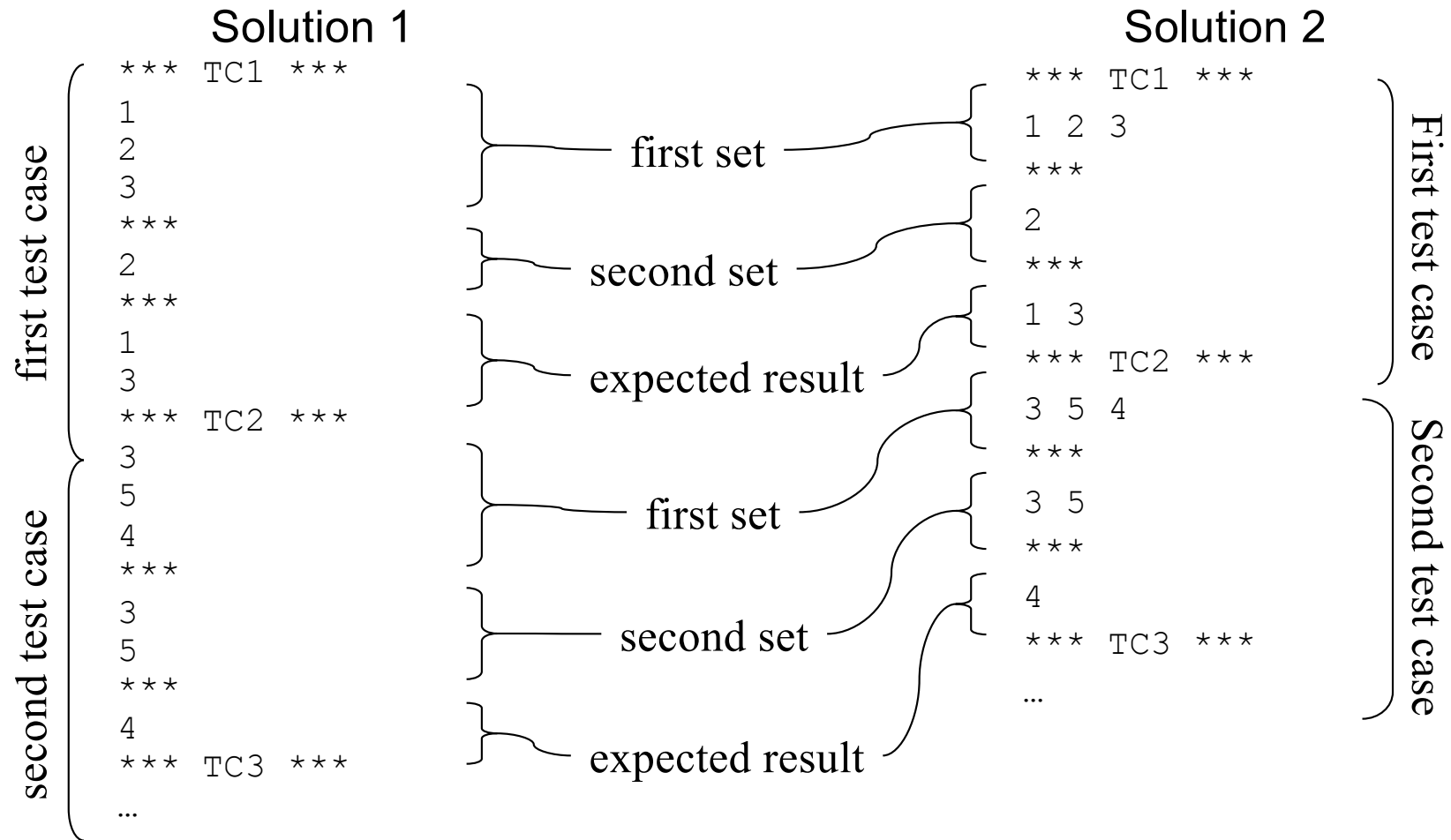
The Oracle—Automation?

- Automating the oracle, i.e., the comparison requires that:
 - We know exactly the expected output (which is the case in the `diff` example)
 - However, sometimes, we do not know exactly what is the result, as (for instance) doing (i.e., implementing) the comparison would correspond to building the function we test
 - We only know approximately the expected output and we can leave with that!
 - Then the driver outputs the test case number when a failure has been detected
 - We only need to know which test cases have failed

The Oracle

- The oracle is included in the driver
 - The expected output must be found in the input text file
 - Format?
- The oracle:
 - Get the output produced by the system under test
 - In our example, the toString method of class OrdSet is called (format of returned string?)
 - Get the expected output from the input text file
 - Performs the comparison
 - In our example, a simple String comparison (equality)

The Oracle



Other Considerations

- Non-determinism
 - What if the behavior of the system under test is not deterministic?
 - Impact on test cases, drivers, stubs, oracles
- Infinite loop because of a fault
 - How do we decide (when automating the execution of test cases) that a test case failed when execution hangs?
- Controllability and observability
 - Lack of mechanisms in the source code to set and get values
 - Putting the system in a given state
 - Getting the state reached after a test case
- The system under test has a GUI
 - The GUI is removed (or bypassed) to facilitate automation during (unit/integration/system) testing.
 - Testing a GUI is a separate task
 - It's not (exactly) system testing

The Oracle Problem

- The *Oracle Assumption*

- The tester is routinely able to determine whether or not the test output is correct

If either of the two following statements holds

- There does not exist an oracle
- It is theoretically possible, but practically too difficult to determine the correct output

Then, the program is said to be *non-testable!*

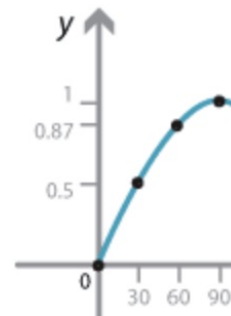
- i.e., if one cannot decide whether or not the output is correct or must expend some extraordinary amount of time to do so, there is nothing to be gained by performing the test!

The Oracle Assumption

- The tester can often state with assurance that a result is incorrect without actually knowing the correct answer.
- Example 1:
 - A program returns a statistical test, a probability.
 - The returned value for some test inputs is $> 1!$ \Rightarrow fault revealed
- Example 2:
 - A program computes the company's total assets and liabilities
 - One test output is \$300! (very likely incorrect)
 - One test output is \$1000! (very likely incorrect)
 - One test output is \$1 000 000. (plausible)
 - Perhaps \$1 100 000 is more plausible, but how can we tell?
 - Can the tester, or a domain expert readily determine that \$1,134,906.43 is correct and \$1,135,627.85 is incorrect?

The Oracle Problem

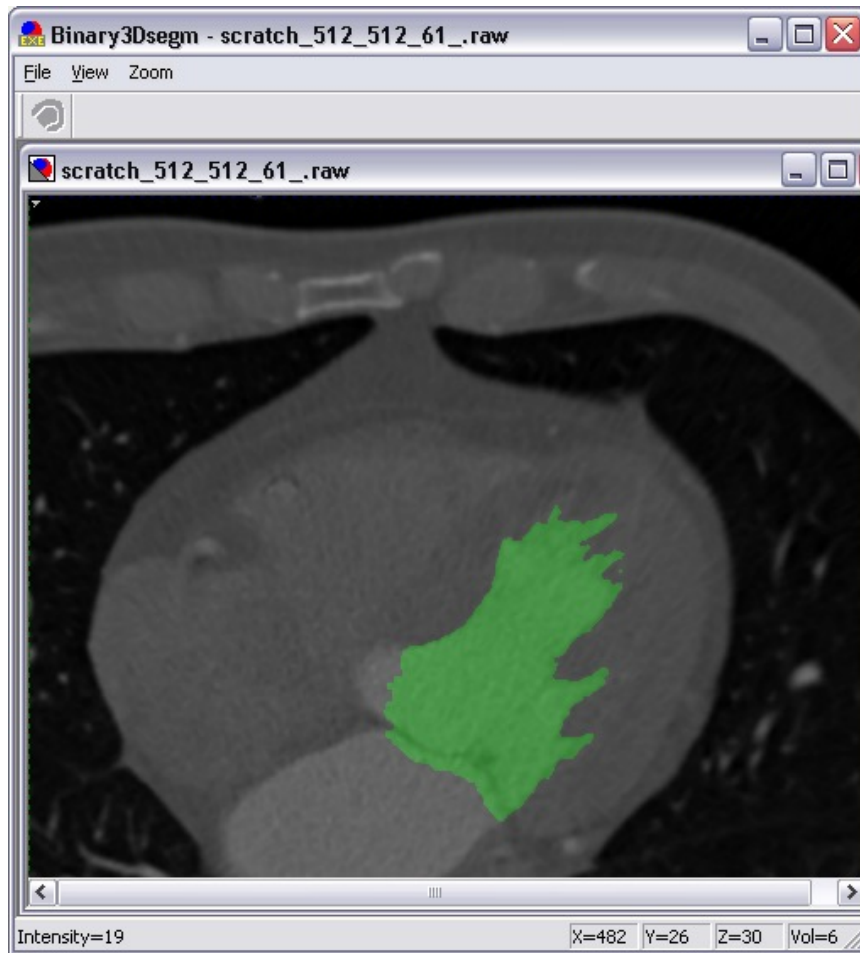
- The tester/expert can usually restrict the range of plausible results.
 - Remember the test of the program that returns the value of a statistical test?
 - Another example: a function that computes $\sin(x)$
 - Test input 42° . What is the expected value? (it is 0.6691)
 - If the returned value is not in range $[-1, 1]$, there is a fault.
 - We know that $\sin(30^\circ)=0.5$ and $\sin(45^\circ)=0.7071$, and that $\sin()$ is strictly increasing in that interval. So if $\sin(42^\circ)$ is not in $[.5, .7071]$ there is a fault.
 - Given that the curve is convex upwards in this interval, a straight line approximation gives a lower bound of 0.6656.



The Oracle Problem

- Another example: medical imaging system
- The algorithm under test is to outline the left ventricle of the heart and compute the volume
 - The information is used by the surgeon for diagnostic purposes
- Verification and validation of the algorithm heavily relies on the expert (the doctor)
 - What is correct, incorrect, almost correct, sufficient for diagnosis?
 - Variations among doctors!
 - Variations over time for a given doctor!

The Oracle Problem



correct



incorrect

Non-Testable Programs

Non-testable program

If either of the two following statements holds

- There does not exist an oracle
- It is theoretically possible, but practically too difficult to determine the correct output

How does this manifest itself?

1. A program written to determine an unknown answer
 - We do not know what a program should return for some given inputs.
 - If the correct answer were known, there would have been no need to write the program
 - E.g., machine learning algorithm
 2. Programs which produce so much output that it is impractical to verify all of it
 - Even a few output data can be too much
 - E.g., Big Data (volume, velocity, variety)
 3. Programs for which the tester has a misconception (difference between ~~actual specification and the one assumed by the tester~~)
-

Non-Tetstable Programs—Solutions?

Categories 1 and 2

- “eyeballing” at the output to see that it “looks okay”, or examining closely parts of the output only (‘simple’ data).
- Run the program on simplified data for which the correctness of the results can be accurately and readily determined.
 - This is in essence what Testing is all about (to avoid exhaustive testing)
 - A program to generate base 2 logarithms might be tested only on numbers of the form 2^n .
- These however (likely) do not exercise boundary values.
- Other strategies
 - Using properties of the problem domain (e.g., computation)
 - E.g., $\sin^2(x) + \cos^2(x) = 1$ (if you know $\cos^2(42)$, then you can compute $\sin^2(42)$)
 - Redo the computation in a different way
 - E.g., $(1-x^2) = (1-x).(1+x) = ((0.5-x)+0.5).(1+x)$
 - Check invariants and contracts (see later discussion)

Non-Tetstable Programs—Solutions? (cont.)

Categories 1 and 2 (cont.)

- Produce a pseudo-oracle (dual coding, in the context of highly critical software)
 - An independently written program intended to fulfill the same specification as the original program
 - Can be a previously written, deemed correct, version of the program (the program has evolved)—Regression Testing
 - The two programs are run on identical sets of inputs.
 - (Automated) Comparison of outputs.
 - Equality
 - Equality with admissible error (e.g., computations)
 - Voting mechanism (fault tolerant computing)
 - Metamorphic testing (see next)
- Overhead!
 - Oracle program should be produced relatively quickly and easily (e.g., reduced specification)

Non-Tetstable Programs

Category 3: misconception

- Correct output deemed incorrect by oracle/tester
 - Time wasted in investigating an inexistent fault
 - Delay in the release
 - The tester/debugger may try to fix the (inexistent) fault, thereby making the program (really) incorrect!
- Incorrect output deemed correct by oracle/tester
 - A fault remains in the program
 - But programs are accepted to have faults
 - Difference between latent fault because exhaustive testing is not feasible, and latent fault because of misconception

Design by Contract—Contracts are Oracles

- Methods/Functions can be specified with pre- and post-conditions; Classes with invariants.
- Example (Java Modeling Language—JML)

```
/*@ requires amount >= 0;
    ensures balance == \old(balance)-amount && \result == balance;
  @*/
public int debit(int amount) {...}
```

- Example (JML)

```
byte[] a; /* The array a is sorted */
/*@ invariant
    (\forall int i; 0 <= i && i < a.length-1; a[i] < a[i+1]);
  @*/
```

Contracts in other languages:

- Native in Eiffel
- Standardization in the work for C++
 - Other solutions exist
- Package available in Python
- Package available in Ada

https://en.wikipedia.org/wiki/Design_by_contract

- This helps software design (adequate responsibilities/expectations)
- Contracts can be checked at runtime automatically
 - E.g., JML compiler

Metamorphic Testing—Procedure

- Correctness is not determined by checking individual concrete outputs
- Correctness is determined by
 - Transforming inputs that lead to outputs that we know should satisfy some relation.
 - Need to identify metamorphic relations (difficult part)
- Procedure
 1. Suppose you know (from the problem domain) that, given input x and corresponding output y (which we do not know), we can identify input p and corresponding output q such that if $R_i(x,p)$ then $R_o(y,q)$.
 - The inputs have a relation $R_i(x,p)$ we know and can control (if we know x , then we can produce p)
 - The outputs have a relation $R_o(y,q)$ we can check (we can compare y and q).
 2. See next ...

Metamorphic Testing—Procedure (cont.)

- Procedure

1. We know $R_i(x,p)$ then $R_o(y,q)$.
2. Then:
 1. generate (possibly randomly) a number of inputs
 - x_1, \dots, x_n .
 2. transform (morph) those inputs according to $R_i(x,p)$:
 - you get p_1, \dots, p_n .
 3. execute program under test with all x_i , and p_i , $i=1 \dots n$:
 - you get y_i and q_i .
 4. if any $R_o(y_i, q_i)$ does not hold, there is a failure.

Metamorphic Testing—Example

- Example: Testing the $\sin()$ function

$$\sin(\pi - x) = \sin(x) \quad \sin(\pi + x) = -\sin(x) \quad \sin(-x) = -\sin(x)$$

- Suppose you have 10 test inputs, x_1, \dots, x_n (possibly randomly generated)
- You can generate 30 other inputs:
 - $\pi - x_1, \dots, \pi - x_n$ $R_i(x, y): y = \pi - x$ $R_o(p, q): q = p$
 - $\pi + x_1, \dots, \pi + x_n$ $R_i(x, y): y = \pi + x$ $R_o(p, q): q = -p$
 - $-x_1, \dots, -x_n$ $R_i(x, y): y = -x$ $R_o(p, q): q = -p$
- Execute 40 tests and compare outputs with each other
(Without knowing the exact output for each of them!)
 - Outputs for $\pi - x_1, \dots, \pi - x_n$ must match that of x_1, \dots, x_n according to $\sin(\pi - x) = \sin(x)$
 - Outputs for $\pi + x_1, \dots, \pi + x_n$ must match that of x_1, \dots, x_n according to $\sin(\pi + x) = -\sin(x)$
 - Outputs for $-x_1, \dots, -x_n$ must match that of x_1, \dots, x_n according to $\sin(-x) = -\sin(x)$

Metamorphic Testing—Example (cont.)

- Example: Testing algorithm that computes shortest path in a graph.
 - If path from n_i to n_j is shortest, then path from n_j to n_i is shortest too.
 - R_i =change direction of path R_o =still shortest path
- Test the program with a (non-trivial) graph and input pairs
 - $(n_1, n_{10}), (n_2, n_9), (n_3, n_5), (n_8, n_4) \dots$
 - Morph
 - Test also with $(n_{10}, n_1), (n_9, n_2), (n_5, n_3), (n_4, n_8) \dots$
 - If
 - outputs for (n_1, n_{10}) and (n_{10}, n_1) do not match
 - outputs for (n_2, n_9) and (n_9, n_2) do not match
 - outputs for (n_3, n_5) and (n_5, n_3) do not match
 - outputs for (n_8, n_4) and (n_4, n_8) do not match
 - ...
- ~~Then there is a failure~~