DATA 520
Lecture 20

Testing and Debugging

Testing and Debugging

Necessary to do early and repeatedly
- saves you time
- save you and others frustration
- saves $$$

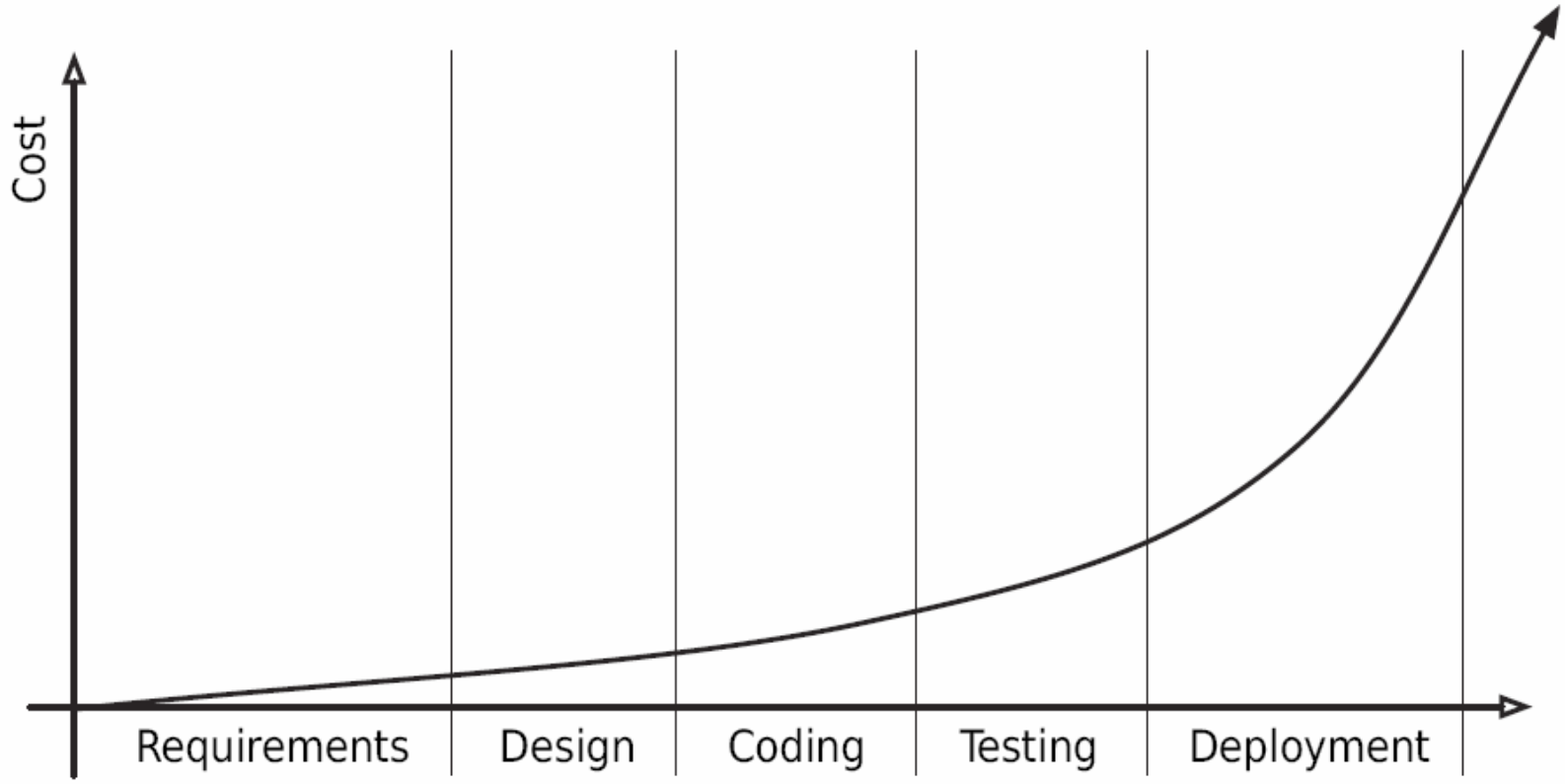

Figure 15—Boehm's curve: the later a bug is discovered, the more expensive it is to fix

Testing and Debugging

Doctest good, needs to be as exhaustive as possible

```
def above_freezing(celsius):
    """ (number) -> bool
    Return True iff temperature celsius degrees is above freezing.
    >>> above_freezing(5.2)
    True
    >>> above_freezing(-2)
    False
    """

return celsius > 0
```

What if celsius = 0?

We will return a boolean (True or False)

Key: What categories of **celsius** values are there?

< 0, > 0, or == 0

```
return celsius >= 0  # would be incorrect
```

Testing and Debugging

Polymorphism (Gries page 283): we can use the same functions on different data types

Example 1:

listL[1:3] # returns a list of 2 things;
stringS[1:3] # returns string of length 2;

Example 2: using for [item] in [itemlist]
- it works for very different items

```
def non_blank_lines(thing):
    """Return the number of nonblank lines in thing."""
    count = 0
    for line in thing:
        if line.strip():
            count += 1

    return count
```

```
>>> for letter in 'How about that?':
        print(letter)
```

- it works for a string, a list of strings, a file, or a web page

Testing and Debugging

Inheritance (pages 284-287): Custom data types can possess (inherit) attributes of other types

```
class Member:
    """ A member of a university. """

    def __init__(self, name, address, email):
        """ (Member, str, str, str) -> NoneType

    self.name = name
    self.address = address
    self.email = email
```

```
class Faculty(Member):
    """ A faculty member at a university. """

    def __init__(self, name, address, email, faculty_num):
    """ (Member, str, str, str, str) -> NoneType
    Create a new faculty named name,
    with home address, email address,
    faculty number faculty_num, and empty list of courses.
    """

    super().__init__(name, address, email)
    self.faculty_number = faculty_num
    self.courses_teaching = []
```

```
class Student(Member):
    """ A student member at a university. """
    def __init__(self, name, address, email, student_num):
    """ (Member, str, str, str, str) -> NoneType
    Create a new student named name,
    with home address, email address,
    student number student_num, an empty list of
    courses taken, and an empty list of current courses.
    """

    super().__init__(name, address, email)
    self.student_number = student_num
    self.courses_taken = []
    self.courses_taking = []
```

Testing and Debugging

Inheritance (pages 284-287):
Custom data types can possess (inherit) attributes of other types

```
class Member:
    """ A member of a university. """

    def __init__(self, name, address, email):
        """ (Member, str, str, str) -> NoneType

class Faculty(Member):

    super().__init__(name, address, email)
    self.faculty_number = faculty_num
    self.courses_teaching = []

class Student(Member):

    super().__init__(name, address, email)
    self.student_number = student_num
    self.courses_taken = []
    self.courses_taking = []
```

Testing and Debugging

We will write test classes that inherit from unittest.TestCase - and use assertEqual

```
# test_above_freezing.py <<<<<<<<------------
import unittest
import temperature

class TestAboveFreezing(unittest.TestCase):  # inheritance
    """Tests for temperature.above_freezing."""
    def test_above_freezing_above(self):
        """Test a temperature that is above freezing."""
        expected = True
        actual = temperature.above_freezing(5.2)
        self.assertEqual(expected, actual,        # assertEqual: test a statement
            "The temperature is above freezing.")

    def test_above_freezing_below(self):
        """Test a temperature that is below freezing."""
        expected = False
        actual = temperature.above_freezing(-2)
        self.assertEqual(expected, actual,
            "The temperature is below freezing.")

    def test_above_freezing_at_zero(self):
        """Test a temperature that is at freezing."""
        expected = False
        actual = temperature.above_freezing(0)
        self.assertEqual(expected, actual,
        "The temperature is at the freezing mark.")

unittest.main() # execute every method beginning with test
```

Testing and Debugging

After running, the output was:

```
temperature module loaded and ready!
the test module has started
__name__ is testmod
Another module is importing me.
...
----------------------------------------------------------------------
Ran 3 tests in 0.003s

OK
```

from temperature.py

from testmod.py

Go back to temperature and change last line to:

```
return celsius >= 0  # would be incorrect
```

- then save and run test_above_freezing.py again

Testing and Debugging

After running with the wrong code, the output was:

```
.F.
======================================================================
FAIL: test_above_freezing_at_zero (__main__.TestAboveFreezing)
Test a temperature that is at freezing.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/Steve9/AppData/Local/Programs/Python/Python36-32/test_above_freezing.py",
line 25, in test_above_freezing_at_zero
    "The temperature is at the freezing mark.")
AssertionError: False != True : The temperature is at the freezing mark.

----------------------------------------------------------------------
Ran 3 tests in 0.004s

FAILED (failures=1)
```

Testing and Debugging

Let's try another one
- notice that this function returns nothing; it mutates a list
- so doctest not practical

```
# sums.py
def running_sum(L):
    """ (list of number) -> NoneType
    Modify L so that it contains the running sums of its original items.
    >>> L = [4, 0, 2, -5, 0]
    >>> running_sum(L)
    >>> L
    [4, 4, 6, 1, 1]
    """

    for i in range(len(L)):
        L[i] = L[i - 1] + L[i]
```

Testing and Debugging

An exhaustive list?

Test Case Description	List Before	List After
Empty list	[]	[]
One-item list	[5]	[5]
Two-item list	[2, 5]	[2, 7]
Multiple items, all negative	[-1, -5, -3, -4]	[-1, -6, -9, -13]
Multiple items, all zero	[0, 0, 0, 0]	[0, 0, 0, 0]
Multiple items, all positive	[4, 2, 3, 6]	[4, 6, 9, 15]
Multiple items, mixed	[4, 0, 2, -5, 0]	[4, 4, 6, 1, 1]

Table 25—Test Cases for running_sum

Testing and Debugging

Testing program part 1

```
# test_sums.py
import unittest
import sums as sums
class TestRunningSum(unittest.TestCase):
    """Tests for sums.running_sum."""

    def test_running_sum_empty(self): # 1
        """Test an empty list."""
        argument = []
        expected = []
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list is empty.")

    def test_running_sum_one_item(self):  # 2
        """Test a one-item list."""
        argument = [5]
        expected = [5]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains one item.")
```

Testing and Debugging

Testing program part 2

```
    def test_running_sum_two_items(self):  # 3
        """Test a two-item list."""
        argument = [2, 5]
        expected = [2, 7]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains two items.")

    def test_running_sum_multi_negative(self):  # 4
        """Test a list of negative values."""
        argument = [-1, -5, -3, -4]
        expected = [-1, -6, -9, -13]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains only negative values.")

    def test_running_sum_multi_zeros(self): # 5
        """Test a list of zeros."""
        argument = [0, 0, 0, 0]
        expected = [0, 0, 0, 0]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains only zeros.")
```

Testing and Debugging

Testing program part 3

```
 def test_running_sum_multi_positive(self):  # 6
        """Test a list of positive values."""
        argument = [4, 2, 3, 6]
        expected = [4, 6, 9, 15]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains only positive values.")

def test_running_sum_multi_mix(self):  # 7
        """Test a list containing mixture of negative values, zeros and positive values."""
        argument = [4, 0, 2, -5, 0]
        expected = [4, 4, 6, 1, 1]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains a mixture of negative
values, zeros and positive values.")

unittest.main()
```

Testing and Debugging

Testing program

```
# test_sums.py
import unittest
import sums as sums
class TestRunningSum(unittest.TestCase):
    """Tests for sums.running_sum."""

    def test_running_sum_empty(self):
        """Test an empty list."""
        argument = []
        expected = []
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list is empty.")

    def test_running_sum_one_item(self):
        """Test a one-item list."""
        argument = [5]
        expected = [5]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains one item.")

    def test_running_sum_two_items(self):
        """Test a two-item list."""
        argument = [2, 5]
        expected = [2, 7]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains two items.")

    def test_running_sum_multi_negative(self):
        """Test a list of negative values."""
        argument = [-1, -5, -3, -4]
        expected = [-1, -6, -9, -13]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains only negative values.")

    def test_running_sum_multi_zeros(self):
        """Test a list of zeros."""
        argument = [0, 0, 0, 0]
        expected = [0, 0, 0, 0]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains only zeros.")

    def test_running_sum_multi_positive(self):
        """Test a list of positive values."""
        argument = [4, 2, 3, 6]
        expected = [4, 6, 9, 15]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains only positive values.")

    def test_running_sum_multi_mix(self):
        """Test a list containing mixture of negative values, zeros and positive values."""
        argument = [4, 0, 2, -5, 0]
        expected = [4, 4, 6, 1, 1]
        sums.running_sum(argument)
        self.assertEqual(expected, argument, "The list contains a mixture of negative values, zeros and positive values.")

unittest.main()
```

Testing and Debugging

Testing program: Run

Failed tests: 3, 4, 6, 7

```
..FF.FF
======================================================================
FAIL: test_running_sum_multi_negative (__main__.TestRunningSum)
Test a list of negative values.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/ten/Documents/test_sums.py", line 32, in test_running_sum_multi_negative
    self.assertEqual(expected, argument, "The list contains only negative values.")
AssertionError: Lists differ: [-1, -6, -9, -13] != [-5, -10, -13, -17]

First differing element 0:
-1
-5

- [-1, -6, -9, -13]
+ [-5, -10, -13, -17] : The list contains only negative values.
```

Oops – first item was modified

Testing and Debugging

Testing program: Run

```
======================================================================
FAIL: test_running_sum_multi_positive (__main__.TestRunningSum)
Test a list of positive values.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/ten/Documents/test_sums.py", line 46, in test_running_sum_multi_positive
    self.assertEqual(expected, argument, "The list contains only positive values.")
AssertionError: Lists differ: [4, 6, 9, 15] != [10, 12, 15, 21]

First differing element 0:
4
10

- [4, 6, 9, 15]
+ [10, 12, 15, 21] : The list contains only positive values.
```

Oops – first item was modified

Testing and Debugging

Testing program: Run

```
======================================================================
FAIL: test_running_sum_one_item (__main__.TestRunningSum)
Test a one-item list.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/ten/Documents/test_sums.py", line 18, in test_running_sum_one_item
    self.assertEqual(expected, argument, "The list contains one item.")
AssertionError: Lists differ: [5] != [10]

First differing element 0:
5
10

- [5]
+ [10] : The list contains one item.
```

Oops – first item was modified

Testing and Debugging

Testing program: Run

```
======================================================================
FAIL: test_running_sum_two_items (__main__.TestRunningSum)
Test a two-item list.
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/ten/Documents/test_sums.py", line 25, in test_running_sum_two_items
    self.assertEqual(expected, argument, "The list contains two items.")
AssertionError: Lists differ: [2, 7] != [7, 12]

First differing element 0:
2
7

- [2, 7]
+ [7, 12] : The list contains two items.
```

Oops – first item item was modified

```
----------------------------------------------------------------------
Ran 7 tests in 0.038s

FAILED (failures=4)
```

Testing and Debugging

Testing program: Run

The ones that passed were empty, only one number, or all zeros

- it was supposed to start with the second element (index 1)

Let's modift the code to fix that

```
# sums.py
def running_sum(L):
    """ (list of number) -> NoneType
    Modify L so that it contains the running sums of its original items.
    >>> L = [4, 0, 2, -5, 0]
    >>> running_sum(L)
    >>> L
    [4, 4, 6, 1, 1]
    """
    for i in range(1, len(L)):  # was for i in range(len(L)): ; now begins with 1
        L[i] = L[i - 1] + L[i]
```

Testing and Debugging

Testing programs

- test an empty list
- test a list with one value
- test a list with two values (the fewest you care about)
- test a list with all positives
- test a list with all negatives
- test a list with mixed numbers (AND zero!)
- test a list with all zeros
- or test as appropriate to your goals

Discrete possibilities (how many POSSIBLE ways of combining values?)

Change the order of things (if they should not be important)

Testing and Debugging

Fix the cause, not just the symptom

Work with known cases – know the correct answer or what you want
Repeat/test the failure – consistent?
(If I change x, it should make the bug 1. better or 2. worse)

Find out WHERE things start to go wrong - `print()` helps
(watch and step in IDEs/debuggers help)

Assume almost nothing (Occam's razor), test from the beginning

Change one thing at a time! (*It is a hypothesis test!*)
(fix, and don't create new bugs)

Keep track of possible and successful corrections, changes: **# comments** help!

Specific error messages: Google can help, but...

Debugging is a process

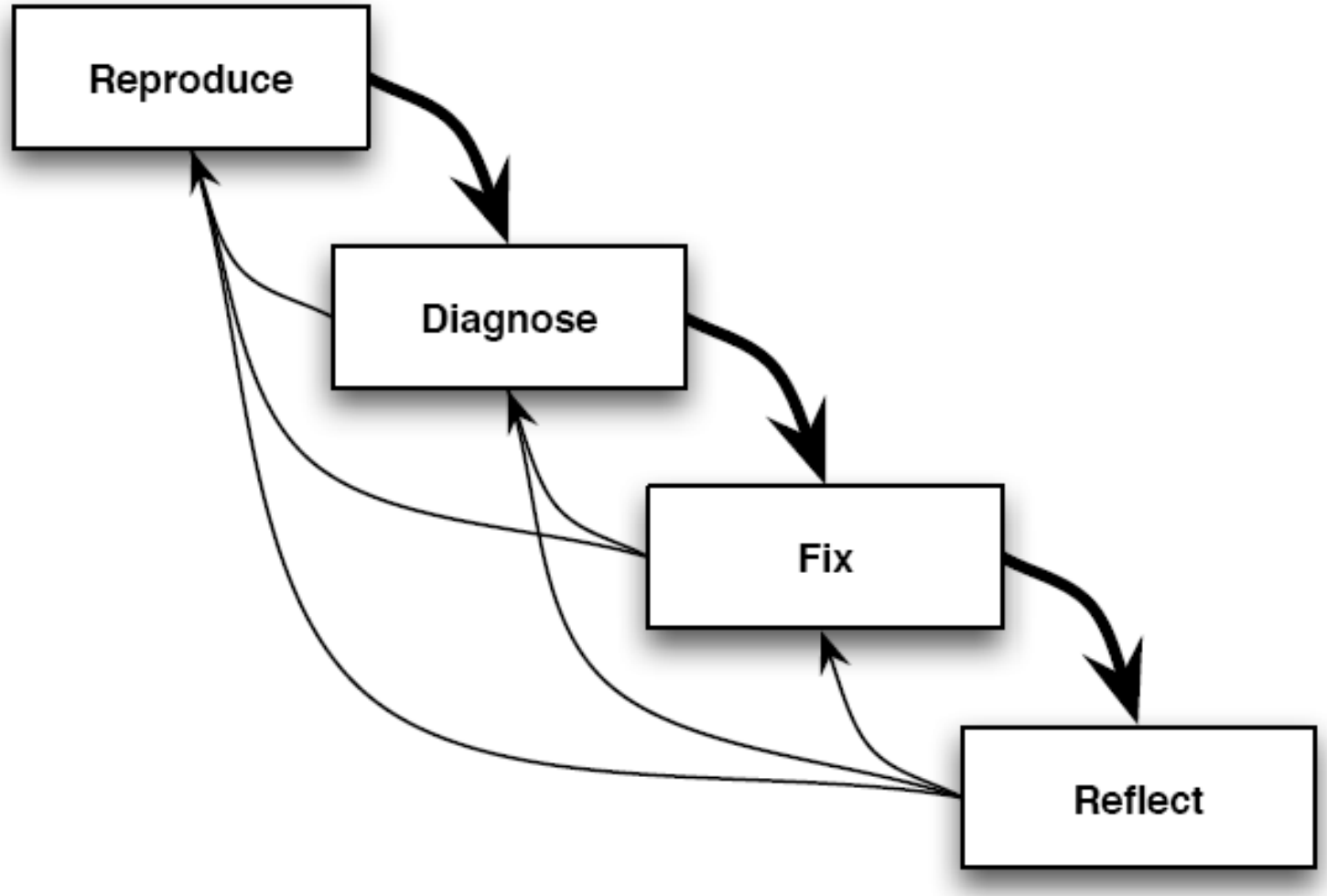

Figure 1.1: Core debugging method

Testing and Debugging

Try to avoid problems in the first place

Try not to use a "kludge" - a "good enough" workaround when programming

See if the operating system can help (APIs - Application Program Interface)
- but beware of unstable APIs

Try to use "flexible" code rather than "hard-wired" code if things may change
(use number of items in a list rather than 25, etc.)

More information on unittest: https://docs.python.org/3/library/unittest.html

Homework 15

Gries 15.7 (page 312)

4, 5, 6

AND:

D. Think of the many ways a list can present to test your program in part B of homework 14 and write a test program using unittest.

14 B. Write a python program to sort a list by magnitude (absolute value). Ties are okay, but negatives should come before positives.

You can use any Python built-in methods (find, min, max, etc.)

Due Wednesday, November 15 before class