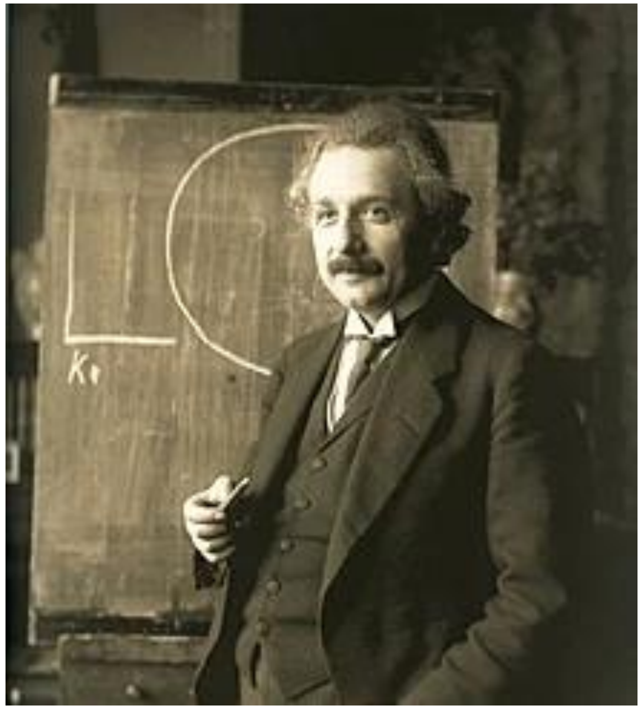
Testing and Debugging

Chapter 8

Testing v Debugging

- **Testing** is the process of running a program to try and ascertain whether or not it works as intended
- **Debugging** is the process of trying to fix a program you already know does <u>not</u> work as intended
  - **p 147**
- These should not be add-ons but designed for
- Test Driven Development
  - Write tests first
  - Build out code until all tests pass



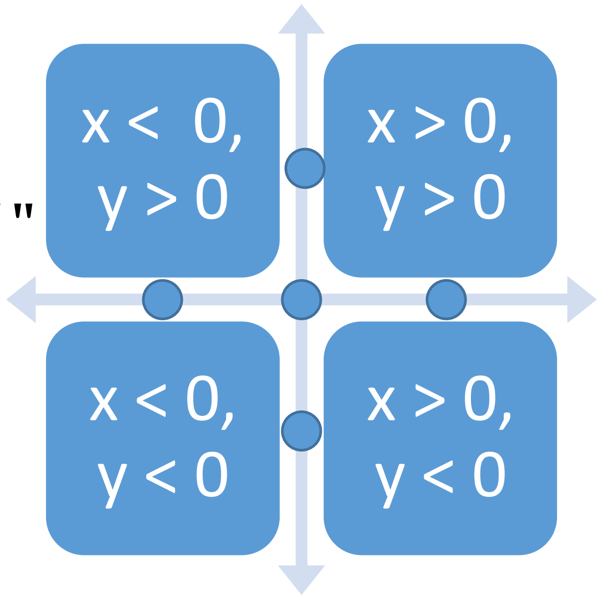
Testing

- “Program testing can be used to show the presence of bugs, never to show their absence” – Edsger Dijkstra
- “No amount of experimentation can ever prove me right, but a single experiment can prove me wrong” – Albert Einstein (credited)
- How did the testing go? “Great we found four bugs!” – Joe Cobb

Test Suite

- Collection of **test cases** that test a set of inputs that will run in a reasonable time that will most likely reveal a bug (defect)
- Define a set of **partitions** from which to draw test cases

```
def isBigger(x, y):
    """Assumes x and y are ints
       Returns True if x is less
       than y and False otherwise."""
```



Test Cases

- Within each partition we may want several test cases.
- How might we want to improve this set of test cases?

Partition	Test Case Inputs	Expected Result
X > 0, Y > 0	1, 2	False
X > 0, Y = 0	1, 0	True
X > 0, Y < 0	1, -1	True
X = 0, Y > 0	0, 1	False
X = 0, Y = 0	0, 0	False
X = 0, Y < 0	0, -1	True
X < 0, Y > 0	-1, 1	False
X < 0, Y = 0	-1, 0	False
X < 0, Y < 0	-1, -1	False

Black Box Testing

- Does not take into consideration the internal logic of a function/module/class
- Can (should) be developed by different team members
  - Reduces the chance of the same assumptions being made by the test and the implementation
- Implementation agnostic
- Simulates how others will use your implementation

Glass Box Testing

- Have access to the implementation of a function
  - Test “magic numbers”
  - Test for **path completeness**

Glass Box Testing Tips (p. 153)

- Exercise both branches of all if statements.
- Make sure that each except clause (see Chapter 9) is executed.
- For each for loop, have test cases in which
  - The loop is not entered (e.g., if the loop is iterating over the elements of a list, make sure that it is tested on the empty list),
  - The body of the loop is executed exactly once, and
  - The body of the loop is executed more than once.
- For each while loop,
  - Look at the same kinds of cases as when dealing with for loops, and
  - Include test cases corresponding to all possible ways of exiting the loop. For example, for a loop starting with

    ```
    while len(L) > 0 and not L[i] == e
    ```
  - Find cases where the loop exits because len(L) is greater than zero and cases where it exits because L[i] == e.

Testing tools

- Stubs and mocks
  - Replace calls to external systems
  - Provide predictable / programmable responses
- Spy
  - Looks into executing code

Comparison of Python Testing Tools

	License	Part of	Category	Category Special feature
Robot	Free software (ASF License}	Python generic test libraries.	Acceptance testing	Keyword-driven testing approach.
PyTest	Free software (MIT License)	Stand alone, allows compact test suites.	Unit Testing	Special and simple class fixture for making testing easier.
unittest	Free software (MIT License)	Part of Python standard library.	Unit Testing	Fast test collection and flexible test execution.
DocTest	Free software (MIT License)	Part of Python standard library.	Unit Testing	Python Interactive Shell for the command prompt and inclusive application.
Nose2	Free software (BSD License)	Carries unittest features with additional feature and plugins.	unittest extension	A large number of plugins.
Testify	Free software (ASF License)	Carries unittest and nose features with additional feature and plugins.	unittest extension	Test discovery enhancement.

https://www.softwaretestinghelp.com/python-testing-frameworks/
Retrieved 29 January 2022

Conducting Tests

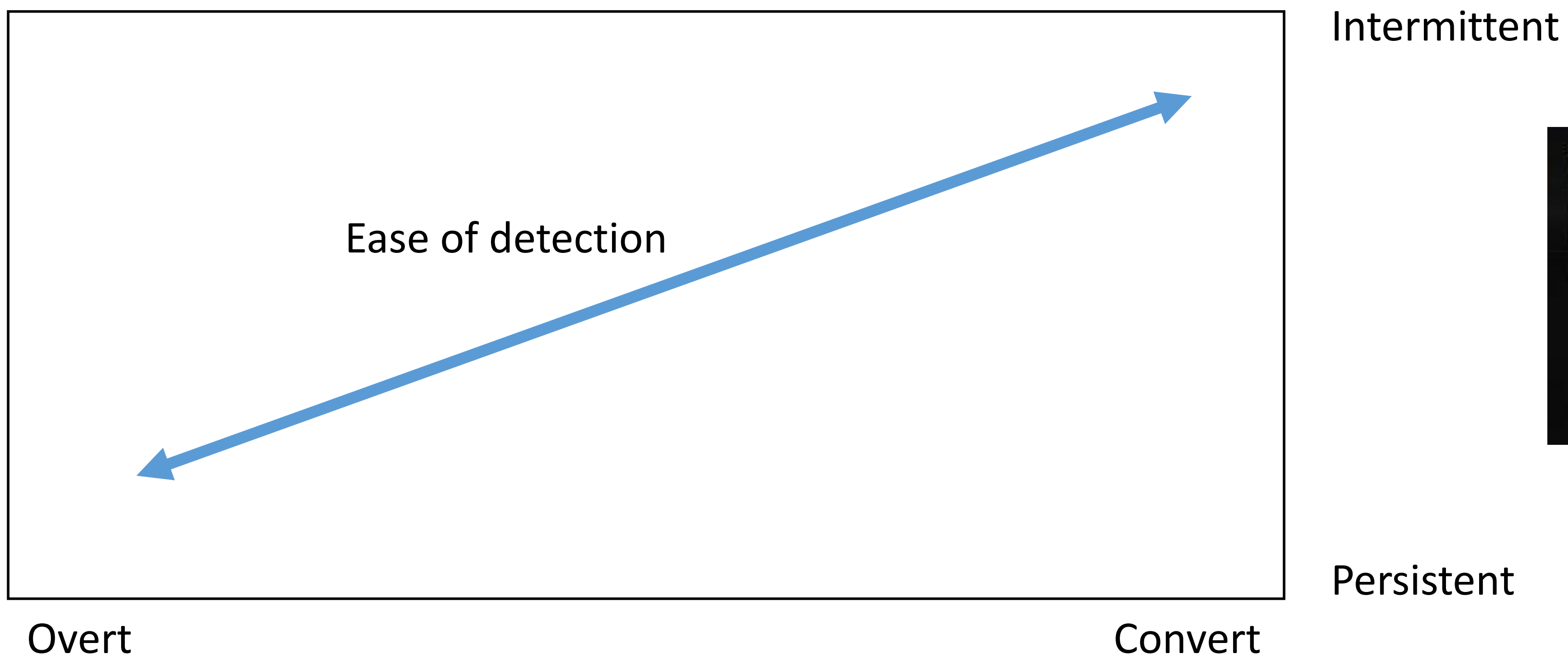
- Unit Tests
  - This is the individual function level Black and Glass Box testing we have been discussion
- Integration Tests
  - Tests a system or sub-system as a whole
  - More challenging to setup environment and predict outcomes
  - Will likely lead to additional unit tests
- Software Quality Assurance (SQA)
  - Separate from Development
  - Often Responsible for Integration Testing
- Automate testing as much as possible

Some more testing

- User Acceptance Testing
- Performance and/or Stress Testing
- Regression Testing
- A/B Testing

Debugging

- Adm. Grace Hopper
- Types of bugs



Bug Classifications

- **Overt & Persistent**
  - Always fails in a predictable manner
  - Easiest to detect and often easiest to fix
- **Overt & Intermittent**
  - Only fails sometimes
  - “It worked for me”
- **Covert**
  - Sneaky little buggers that are hard to detect and often hard to fix

Learning to debug
Diagnosis

- What type of error is this; crash, hang, wrong result
- What can cause that type of error?
- Review the tests
- Review the code
- Create a hypothesis

Learning to debug
Design an experiment

- Create new tests
  - Replicate the defect
  - Test around the error
- Divide and conquer
  - Work from the input end
    - Next week will look at exception handling
  - Determine **breakpoints**
  - Inspect variables, function results
    - Debug by printing
    - Interactive debugging

Python debugger

```
import pdb

pdb.set_trace()
```

- Some commands
    - `h(elp) [command]`
    - `s(tep)`
    - `n(ext)`
    - `c(ontinue)`
    - `p(rint) [expression]`
    - `q(uit)`
- https://docs.python.org/3.7/library/pdb.html?highlight=pdb

When the going gets tough

- Look for the *usual suspects*
- Not – why isn’t it doing the “right” thing, but why is it doing the “wrong” thing?
- Think different
- Explain it to somebody
- Don’t believe the documents
- Write the docs
- Live to fight another day

  pp 164

When you find “the” bug

Is this the only problem or only a problem

Don’t just correct the error – improve the solution