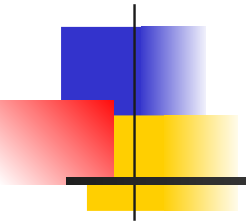


Intro to Testing

James Brucker

Many Kinds of Software Testing

Software testing is critical!

- Test requirements - consistent? unambiguous?
- Test software specification - is it consistent with the requirements? Satisfy all the requirements?
- Unit Testing - test individual methods and functions
- Integration Testing
- End-to-End or Functional Testing
- Acceptance Testing
- Usability Testing

Why Test?

1. *Saves time!*

   - *Testing is faster than fixing "bugs".*

2. *Testing finds more errors than debugging.*

3. *Prevent re-introduction of old faults (regression errors).*

   Programmers often recreate an error (that was already fixed) when they modify code.

4. *Validate software: does it match the specification?*

Psychological Advantages

- *Keeps you **focused** on current task.*
- *Increase **satisfaction**.*
- ***Confidence** to make changes.*

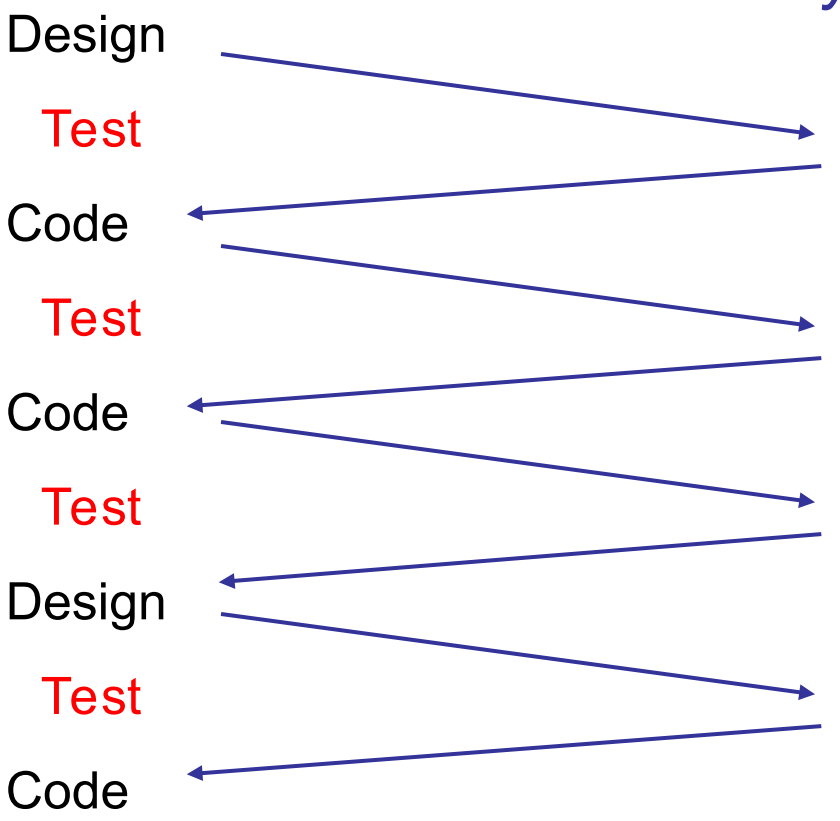
Test-Driven Development (TDD)

*Write the **tests first** ... what the code should do.*
Then write code that passes the tests.

Testing is part of development

Agile Development Practices

- *Test early.*
- *Test continually!*



When To Test?

*- Test **while** you are writing the source code*

***- Retest** whenever you modify the source code*

The Cost of Fixing "faults"

Discover & fix a defect early is much cheaper (100X) than to fix it after code is integrated.

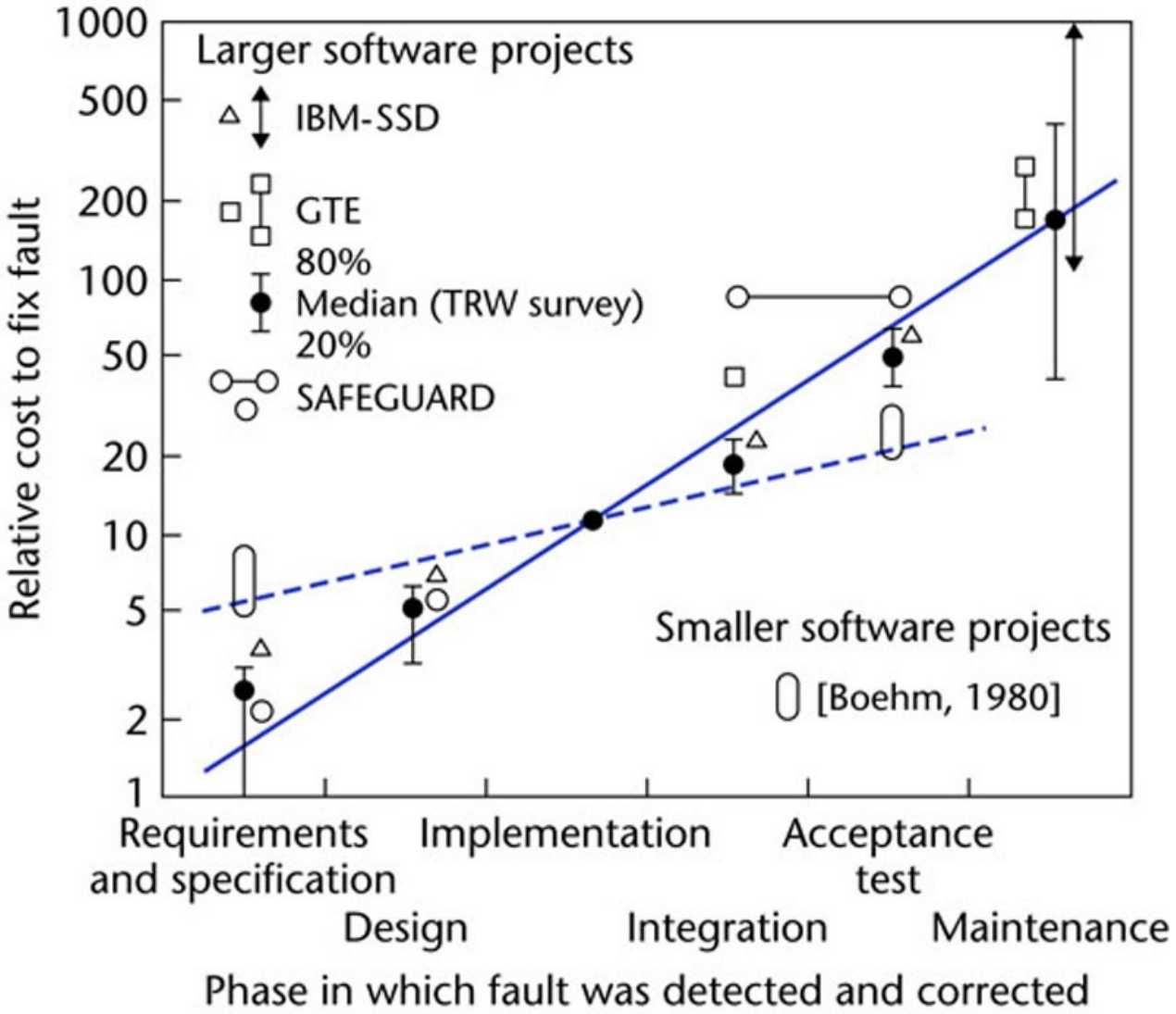


Figure 1.5

What to Test?

In unit testing, we test functions or methods in classes.

How to Test?

We can not test all possible input / output.

- Divide input into categories or sets.
- Discover "rules" that apply to different sets of input.
- Test a few samples from each set, category, or class.
  - Test boundary values.
  - Test "typical" values.
  - Test "extreme" values.
  - Test impossible values.
  - Try to make the code fail.

Example: gcd(a,b)

`gcd(a:int, b:int)` = greatest common divisor of a & b

`gcd(24,30)` -> 6

`gcd(3, 7)` -> 1 (no common factors)

Rule: gcd is always positive

`gcd(80,-15)` -> 5

`gcd(-7,-3)` -> 1

Rule: gcd involving zero is positive

`gcd(8,0)` -> 8

`gcd(0,-8)` -> 8

Defining Test Cases

Test Case	Example Arguments
Two positive ints with common factor	(30, 35), (48, 20), (36, 999)
Two int with no common factor	(1, 50), (50, 3), (370, 999), (1,1)
One or both args are negative	(-30,45), (72,-27), (-1,-2)
One or both args zero	(99, 0), (0,-33), (0,0)
Extreme case to test algorithm efficiently terminates	(123*123457890123, 123*789012345890)

Python unittest

```
import unittest

class TestGcd(unittest.TestCase):
    def test_gcd_positive_values(self):
        self.assertEqual(6, gcd(30,35))
        self.assertEqual(4, gcd(48,20))

    def test_gcd_involving_zero(self):
        self.assertEqual(1, gcd(0,0))
        self.assertEqual(99, gcd(0,99))
        self.assertEqual(1, gcd(1,0) )
```

FIRST - guide for good tests

Fast

Independent - can run any subset of tests in any order

Repeatable - always get same result

Self-checking - test knows if it passed or failed

Timely - written at same time as the code to test

Stack Example

- Stack implements a stack data structure.
- Has a fixed capacity and methods shown below.
- Throws StackException if you do something stupid.

Stack<T>
+ Stack(capacity)
+ capacity( ): int
+ size( ): int
+ isEmpty( ): boolean
+ isFull( ): boolean
+ push(T): void
+ pop( ): T
+ peek( ): T

What to Test?

Border Case:

Stack with capacity 1

1. no elements in stack
   capacity() is 1
   isEmpty() -> true
   isFull() -> false
   size() -> 0
   peek() returns ???
2. push one element on stack
   isEmpty() -> false
   isFull() -> true
   size() -> 1
3. can peek()?
   push one element
   peek() returns element
   stack does not change
4. push element, peek it, then pop
   pop -> returns same object
   test all methods
   idea: a helper method for all tests of an empty stack or full stack

Test for Methods

push( )

Hard to test by itself!

Need to use peek( ), pop( ), or size( )

to verify something was pushed.

1. Stack of capacity 2.

push(x)

verify size=1 peek()==x, not full, not empty

push(y)

verify again

pop(y)

push(x) - should have 2 items both == x

References

`unittest` in Python Library

- Search for "unittest" on main page of Library
- You should have this installed on your own computer

Test-Driven Development in Python, 2E

- "the testing book" for Python
- create & test a Django app using TDD

Hitchhiker's Guide to Python Testing

- short introduction to several testing tools
- `https://docs.python-guide.org/writing/tests/`