



# **Software Testing**

Juyoung Yang, Arogya Kharel didwndud3299@kaist.ac.kr, akharel@kaist.ac.kr

Web Engineering and Service Computing Lab, Korea Advanced Institute of Science and Technology (KAIST)

2025. 05. 28~30





### Contents

- 1. Introduction to Software Testing
- 2. Unit Test
- 3. Automated Test Case Generation
- 4. Automated Continuous Integration





# Introduction to Software Testing

## **Software Testing**

An investigation conducted to provide stakeholders with information about the quality of the software under test (SUT)

- Software quality attributes
  - Correctness (Functionality)
     We will focus on this!
  - ☐ The software behaves exactly as intended according to its specifications
  - Performance
  - ☐ The software meets some performance-related expectations (execution time, network throughput, memory usage)
  - Usability
  - □ The software is easy to learn and use (intuitive user interfaces)
  - Reliability
  - The software performs well under various conditions and for a specific period



## Why is Software Testing Important?

#### Even a small & trivial bug can cause big damage

#### Ariane 5<sup>[1]</sup>

- The Ariane 5 rocket, developed by the European Space Agency, exploded just 40 seconds after launch
- The failure was caused by an integer overflow
- This small error resulted in the loss of the rocket and cargo worth around 600 million euros





## Why is Software Testing Important?

#### Even a small & trivial bug can cause big damage

- Toyota Recall<sup>[2]</sup>
  - 625,000 hybrid cars recalled in 2015 due to a software glitch in the braking system





#### Introduction

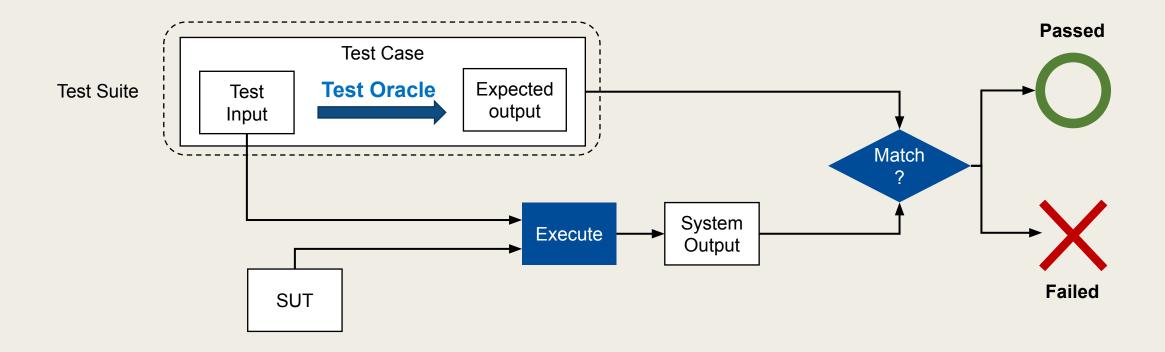
## **Software Testing Terminology**

- **Test Input**: a set of input values that are used to execute the given program
- **Test Oracle**: a provider of the expected output based on the test input
- **Test Case:** test input + expected output
- Test Suite: a collection of test cases



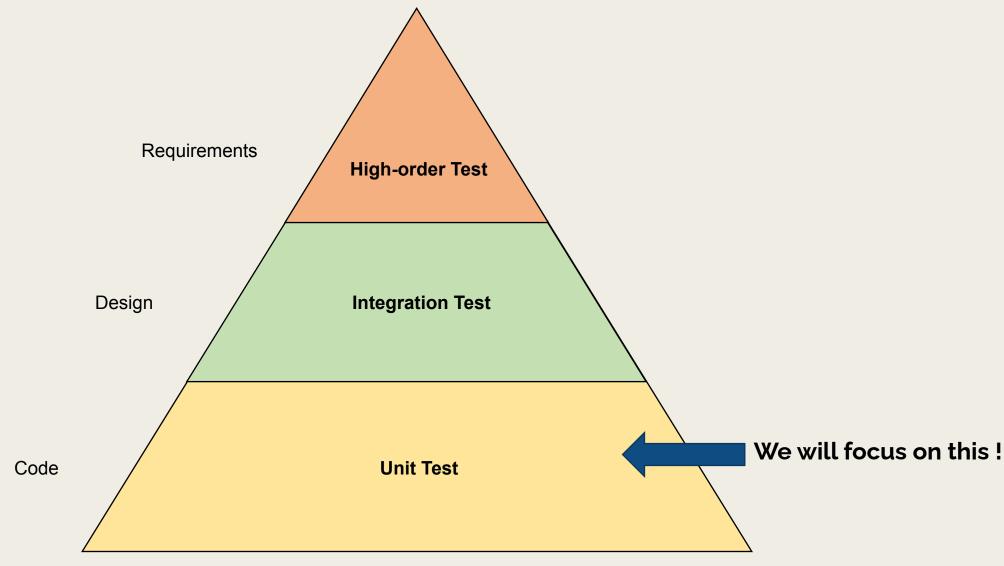
## **Process of Software Testing**

Check whether the system's output and the expected output are the same





## **Software Testing Hierarchy**







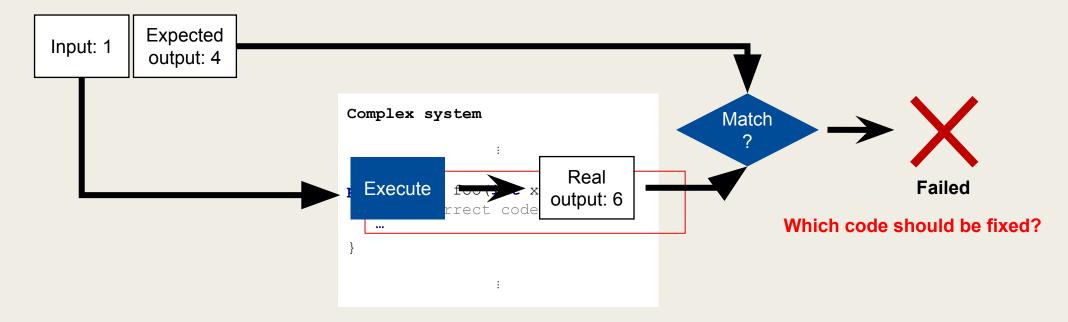


# **Unit Testing (Python)**

### Why Unit Test?

Test each unit (function, class) of the software to check whether they satisfy the requirements

□ Detect bugs early and ensure each unit works correctly in isolation





### **SUT in This Tutorial**

Basic requirements of a car

A car tracks its driving time, distance, and speed.

A car can change its speed.

A car should be able to return its current speed and average speed.

- Based on the requirements, the following code can be written
- □ car.py (See Appendix A)

```
class Car:
    def __init__(self, speed=0):
        self.speed = speed
        self.odometer = 0
        self.time = 0

    def change_speed(self, change):
        ...
    def get_current_speed(self):
        ...
    def average_speed(self):
        ...
```



## **Writing and Running Test Cases**

- Based on the SUT, unit test code can be written 

  test\_car.py (Appendix B)
  - Specify test input to get the real output
  - Specify the expected test output
  - Compare the real output and expected output

```
import pytest
import random
from car import Car

@pytest.fixture
def car():
    return Car()

def test_initial_speed(car):
    assert car.speed == 0
```

- Run test cases
  - pytest <test\_file>
  - Ex) pytest test\_car.py



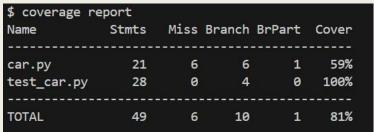
## **Checking Test Coverage**

#### Test coverage

- How much of the source code is executed by the test suite
  - Statement coverage
  - Branch coverage
- High coverage generally leads to more reliable and robust software
- However, 100% coverage does not guarantee 100% correctness
- ☐ Coincidental correctness: quality of test cases also matters

#### Check test coverage

- coverage run -branch -m pytest test car.py
- · coverage report





#### **Unit Testing**

### **Tutorial Setting**

Install Python 3.10.11

https://www.python.org/downloads/release/python-31011/

Windows installer (64-bit) Windows Recommended a55e9c1e6421c84a4bd8b4be41492f51 27.7 MB SIG .sigstore

- Confirm git installation
  - Open Powershell
  - Try git version
- If error, install git

https://github.com/git-for-windows/git/releases/tag/v2.49.0.windows.1

Git-2.49.0-64-bit.exe

726056328967f242fe6e9afbfe7823903a928aff577dcf6f517f2fb6da6ce83c



#### **Unit Testing**

### **Tutorial Setting**

 Fork the tutorial repository from the URL below <u>https://github.com/kaist-webeng/software-testing-exercise</u>

- Clone your forked repository
  - git clone <a href="https://github.com/<your ID>/software-testing-exercise">https://github.com/<your ID>/software-testing-exercise</a>
- Install the required python packages (required python version: 3.10)
  - pip install –r requirements.txt



### **Challenges of Generating Test Cases**

#### Exhaustive Testing

- Can we test all possible test inputs?
  - Ex) Password Checker: checks 8-character passwords using only uppercase letters  $(A-Z) \square$  There are 26<sup>8</sup> = 208,827,064,576 possible combinations
- All test inputs need to be executed & the result should be compared with expected result 

   prohibitive cost

#### Small & effective test suite

- Testing allows only a sampling of an enormously large program input space
- The difficulty lies in how to come up with effective sampling



### **Challenges of Generating Test Cases**

#### Quality of test suite

- A high-quality test suite ensures high coverage and includes edge cases that can reveal the faults
- However, manually writing such test cases is time-consuming and error-prone
- Automated test generation tools
  - Create high-coverage, effective test cases without human effort







### **Automated Test Case Generation**

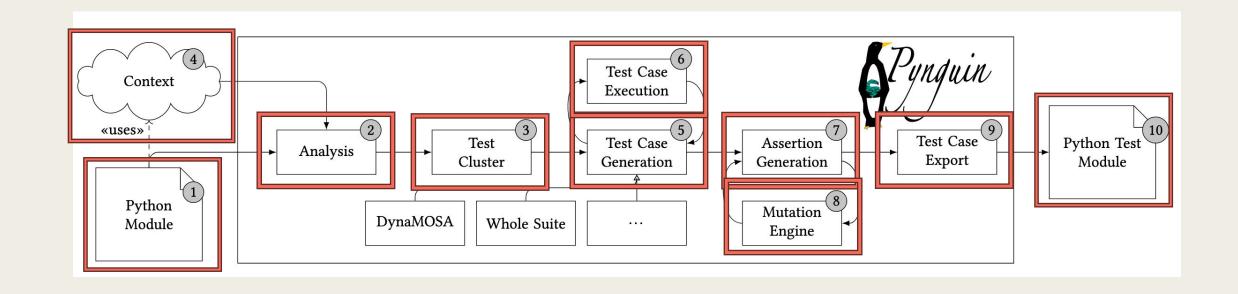
### What is Pynguin?

- PYthoN General UnIt Test generator[1, 2]
  - A tool that allows developers to generate unit tests automatically.
- Mature tools exist for statically typed languages like Java
  - No comprehensive tool existed for dynamically typed languages like Python
- Pynguin steps in as the first tool to enable fully-automated generation





## **Execution Steps of Pynguin**





### Going Back Inside Our Car

Pynguin requires setting env var<sup>[3]</sup>:

```
export PYNGUIN_DANGER_AWARE=x
```

 Provide pynguin with source and output paths:

```
$ pynguin --project-path .
--output-path pynguin-results
--module-name car
```

Check your generated test file!

```
class Car:
    def __init__(self, speed=0):
        self.speed = speed
        self.odometer = 0
        self.time = 0

    def change_speed(self, change):
        ...
    def get_current_speed(self):
        ...
    def average_speed(self):
        ...
```

```
$ coverage report

Name Stmts Miss Branch BrPart Cover

car.py 21 6 6 1 59%

test_car.py 28 0 4 0 100%

TOTAL 49 6 10 1 81%
```

#### **Test Generation**

# **Pynguin Results**

```
1 # Test cases automatically generated by Pynguin (https://www.pynguin.eu).
   # Please check them before you use them.
    import pytest
    import car as module_0
                                    The test is expected to fail!
    @pytest.mark.xfail(strict=True)
   def test_case_0():
       bool_0 = False
       car_0 = module_0.Car()
10
       assert car_0.speed = 0
11
       assert car_0.odometer = 0
       assert car_0.time = 0
13
       var_0 = car_0.change_speed(bool_0)
       assert car_0.time = 1
       var_0.step()
    def test_case_1():
       none_type_0 = None
       car_0 = module_0.Car(none_type_0)
21
       assert car_0.odometer = 0
       assert car_0.time = 0
       with pytest.raises(Exception):
           car_0.average_speed()
27
    def test_case_2():
29
       car_0 = module_0.Car()
       assert car_0.speed = 0
       assert car_0.odometer = 0
       assert car_0.time = 0
```

\$ coverage repo Name S	rt tmts	Miss	Branch	BrPart	Cover
car.py test_car.py	21 28	6 0	6 4	1 0	59% 100%
TOTAL	49	6	10	1	81%

VS

Name	Stmts	Miss	Branch	BrPart	Cover
car.py test_car.py	21 75	0 U	<b>]</b> 6	 0 0	100% 100%
TOTAL	96	0	6	0	100%





# **Automated Continuous Integration**

## **Automated Continuous Integration**

- Continuous Integration (CI): a development practice where developers
  frequently integrate code into a shared repository
- Automated CI enables that code changes are automatically built, tested, and validated each time they are committed
- ☐ It is essential for fast and error-free integration of the code

#### CI tools

GitHub Actions



Actions

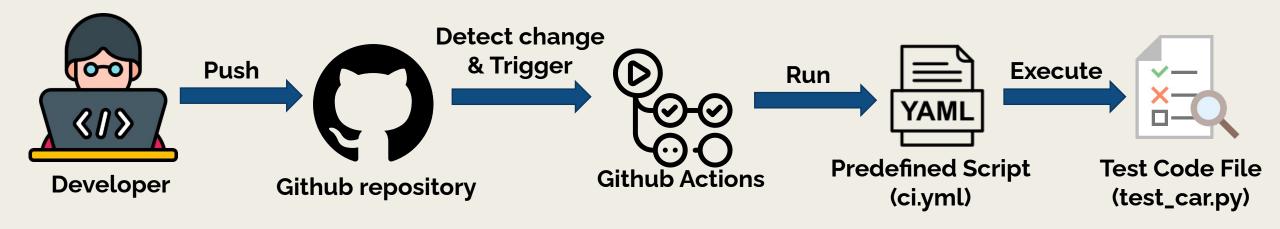
Jenkins





### **Github Action**

- When a developer pushes the changed code, GitHub Actions automatically run a predefined script that executes test code
- ☐ Our predefined script: ci.yml (See Appendix F)





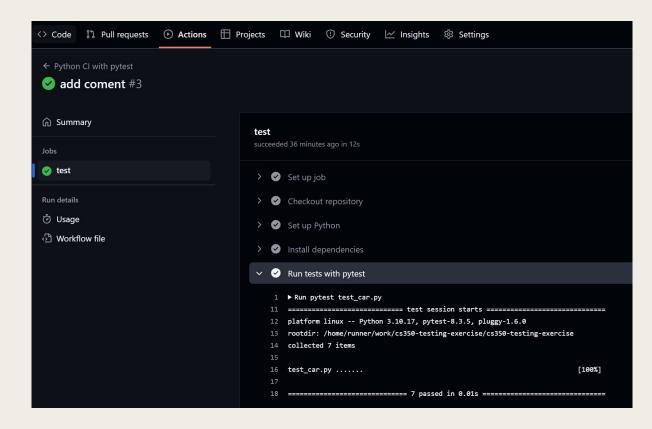
#### CI

### **Github Action**

Add your CI script (ci.yml) to below path

#### <root directory>/.github/workflows

- Change your source code (car.py)
- Push your new commit
- Check your predefined script for CI is automatically executed









# Thank you

Juyoung Yang, Arogya Kharel didwndud3299@kaist.ac.kr, akharel@kaist.ac.kr

Web Engineering and Service Computing Lab, Korea Advanced Institute of Science and Technology (KAIST)

2025. 05. 28~30





## Appendix

- A. Source Code car.py
  - B. Sample Test Code test\_car.py
  - C. Source Code lift.py
  - D. Sample Test Code test\_lift.py
  - C. Generated Test Code pynguin-results/test\_car.py

### Source Code - car.py

```
class Car:
    def __init__(self, speed=0):
        self.speed = speed
        self.odometer = 0
        self.time = 0
   def change_speed(self, change):
        if change < 0:</pre>
           if self.speed + change < 0:</pre>
                self.speed = 0
           else:
                self.speed += change
        else:
            self.speed += change
        self.step()
    def get_current_speed(self):
        return self.speed
    def step(self):
        self.odometer += self.speed
        self.time += 1
    def average_speed(self):
        if self.time == 0:
            raise Exception("Divide by 0! Car did not move!")
        else:
            return self.odometer / self.time
```



### Sample Test Code - test\_car.py

```
import pytest
import random
from car import Car
@pytest.fixture
def car():
    return Car()
def test initial speed(car):
    assert car.speed == 0
def test initial odometer(car):
    assert car.odometer == 0
def test_initial_time(car):
    assert car.time == 0
```

```
# Tests for changing speed
def test_change_speed(car):
    car.change_speed(7)
    assert car.speed == 7
def test_multiple_speed_change(car):
    total_change = 0
    random.seed(42)
    for _ in range(3):
        change = random.randint(0, 10)
        total_change += change
        car.change_speed(change)
    assert car.speed == total_change
# Tests for getting current speed
def test_get_current_speed_from_start(car):
    assert car.get current speed() == 0
def test_get_current_speed_after_change(car):
    car.change_speed(27)
    assert car.get_current_speed() == 27
```



### Sample Test Code - lift.py

```
class Lift:
  def init (self, highest floor, max riders =10):
       self.top floor = highest floor
      self.current floor = 0
       self.capacity = max riders
  def get top floor (self):
       return self.top floor
  def get current floor (self):
       return self.current floor
  def get capacity (self):
       return self.capacity
  def get num riders (self):
  def is full(self):
       return self.num riders == self.capacity
```

```
def add riders (self, num entering):
    if self.num riders + num entering <= self.capacity:</pre>
        self.num riders += num entering
        self.num riders = self.capacity
def go up(self):
    if self.current floor < self.top floor:</pre>
        self.current floor += 1
def go down(self):
    if self.current floor > 0:
        self.current floor -= 1
def call(self, floor):
    if 0 <= floor <= self.top floor:</pre>
        while self.current floor != floor:
            if floor > self.current floor:
                self.go up()
            else:
                self.go down()
```



### Sample Test Code – test\_lift.py

```
import pytest
from lift import Lift
  lift = Lift(5)
  assert lift.get top floor() == 5
   assert lift.get current floor() == 0
  assert lift.get num riders() == 0
   assert not lift.is full()
  lift = Lift(5)
  lift.add riders(5)
   assert lift.get num riders() == 5
  assert not lift.is full()
  lift = Lift(3)
  lift.go up()
  lift.go up()
  lift.go up() # should be at top
  lift.go up() # should stay at top
   assert lift.get current floor() == 3
```



#### **Appendix E**

### Generated Test Code - pynguin-results/test\_car.py

```
# Test cases automatically generated by Pynguin
(https://www.pynguin.eu).
# Please check them before you use them.
import pytest
import car as module 0
def test case 0():
   float 0 = -13.59933065467643
   car 0 = module 0.Car()
   assert car 0.speed == 0
   assert car 0.odometer == 0
   assert car_0.time == 0
   var_0 = car_0.change_speed(float_0)
   assert car 0.time == 1
@pytest.mark.xfail(strict=True)
def test case 1():
   bool 0 = False
   car 0 = module 0.Car()
   assert car 0.speed == 0
   assert car 0.odometer == 0
   assert car 0.time == 0
   var 0 = car 0.change speed(bool 0)
   assert car 0.time == 1
   var 0.step()
def test case 2():
   float 0 = -35.84348955890579
   car 0 = module 0.Car()
   assert car_0.speed == 0
   assert car 0.odometer == 0
   assert car 0.time == 0
   var 0 = car 0.change speed(float 0)
   assert car_0.time == 1
   car 1 = module 0.Car()
   assert car 1.speed == 0
   assert car 1.odometer == 0
   assert car 1.time == 0
   car 2 = module 0.Car()
   assert car 2.speed == 0
   assert car_2.odometer == 0
   assert car 2.time == 0
   with pytest.raises(Exception):
       car 2.average speed()
```

```
@pytest.mark.xfail(strict=True)
def test case 3():
   float_0 = -12.76562758095508
   car 0 = module 0.Car()
   assert car 0.speed == 0
   assert car 0.odometer == 0
   assert car 0.time == 0
   var 0 = car 0.change speed(float 0)
   assert car 0.time == 1
   car 1 = module 0.Car()
   assert car_1.speed == 0
   assert car 1.odometer == 0
   assert car 1.time == 0
   var_1 = car_0.average_speed()
   assert var 1 == pytest.approx(0.0, abs=0.01, rel=0.01)
   var 0.change speed(car 1)
def test case 4():
   car 0 = module 0.Car()
   assert car 0.speed == 0
   assert car 0.odometer == 0
   assert car 0.time == 0
@pytest.mark.xfail(strict=True)
def test case 5():
   car 0 = module 0.Car()
   assert car_0.speed == 0
   assert car 0.odometer == 0
   assert car 0.time == 0
   var 0 = car 0.step()
   assert car_0.time == 1
   var 1 = car 0.get current speed()
   assert var 1 == 0
   car_0.change_speed(car_0)
```

```
@pytest.mark.xfail(strict=True)
def test case 6():
   car_0 = module_0.Car()
   assert car 0.speed == 0
   assert car 0.odometer == 0
   assert car 0.time == 0
   var 0 = car 0.step()
   assert car 0.time == 1
   var 0.step()
@pytest.mark.xfail(strict=True)
def test case 7():
   float_0 = -0.23206040601398698
   car 0 = module 0.Car()
   assert car 0.speed == 0
   assert car 0.odometer == 0
   assert car 0.time == 0
   var 0 = car 0.change speed(float 0)
   assert car 0.time == 1
   bool 0 = True
   var 1 = car 0.change speed(bool 0)
   assert car 0.speed == 1
   assert car 0.odometer == 1
   assert car 0.time == 2
   var 2 = car 0.get current speed()
   assert var 2 == 1
   var 3 = car 0.change speed(float 0)
   assert car_0.speed == pytest.approx(0.767939593986013, abs=0.01, rel=0.01)
   assert car 0.odometer == pytest.approx(1.767939593986013, abs=0.01, rel=0.01)
   assert car 0.time == 3
   car 0.change speed(var 1)
```



### CI Script - ci.yml

```
name: Python CI with pytest
 push:
 pull request:
jobs:
 test:
   runs-on: ubuntu-latest
    - name: Checkout repository
       uses: actions/checkout@v4
     - name: Set up Python
       uses: actions/setup-python@v5
         python-version: "3.10"
     - name: Install dependencies
        python -m pip install --upgrade pip
        pip install -r requirements.txt
     - name: Run tests with pytest
        pytest test car.py
```







# **EOF**