CS 350 – Software Engineering

November 3, 2017

Week 11, Class #32

Unit Testing

Mark Seaman

MWF – 10:00-11:30 - Kepner 0095F

This Week – Test

- ✧ Monday, 10-30
  - Lecture – Types of Test
- ✧ Wednesday, 11-1
  - Lecture – Unit Tests
- ✧ Friday, 11-3
  - Lecture – System Tests

Last Week

Code

Test

Next Week

Dev Ops

Exercises

- Version Control
  - Markdown Exercise 10/20
  - Github Login 10/30

- Development
  - Design Plan 10/23
  - Development Exercise 10/25
  - Pair Programming Exercise 10/30
  - Unit Test Exercise 11/3

Development Exercise

- ✧ Exercise Instructions
  - https://github.com/UNC-CS350
  - Repo - CS350
  - Exercises
    - Unit_Test.md
    - Results/Programming_Pairs.md

- ✧ Exercise Results
    - Student-id/unit_test.py

Types of Testing

- ✧ Unit testing – isolated features
- ✧ Component testing – features in context
- ✧ System testing – end-to-end

Development travels at the speed of test

Acceptance Testing

What is acceptance testing?

✧ Traditional vs. Modern

✧ Focus on automation for repeatable results

Requirements

✧ Say it with code

✧ Execute the code

Continuous Integration

- Every commit gets tested
- System tests – test of integration
- Unit tests – isolate failures

Measure Success

Metrics Drive Success – you get what you measure

- ✧ Time to catch defects
- ✧ Coverage
- ✧ Time to fix
- ✧ Issues caught
- ✧ Repeat offenders
- ✧ Regression tests

Personal Story

Managing several hundred distributed server

Wide variety of functional requirements

Constant change

Only person to manage the system

Hammer Test

✧ Every test produces text

✧ Know correct answers are stored

✧ Differences are shown

✧ Approve current answer

Hammer Test

```
def test_num_files ():
    n = shell (‘find ..’)
    assert (800 < n and n <1000)

def test_system_files ():
    print (shell (‘find ..’))

def test_num_lines ():
    n = shell (‘cat *’)
    assert (8000 < n and n < 9000)
```

Hammer Test – UNC-CS350

- ✧ Has local files
- ✧ Remote access is working
- ✧ Has specific files
- ✧ Number of lines in Exercises
- ✧ Student results
- ✧ Push and pull
- ✧ Check on web server results

See you Monday

✧ Complete Unit Test Exercise

X

- Y
  - Z

Unit Testing Run Function

```
test_all
    test_thing_1 ()
    test_thing_2 ()
    test_thing_3 ()
    test_thing_4 ()
test_thing_1
    assert (thing (object), answer)
test_all()
```

Unit Testing

- ✧ Start with Classes

  User class

    first

    last

    email

    name()

- ✧ Test each feature with one test case
  - Keep it simple
  - Run all tests

Simple Unit Test

```
# Make sure that tests run
test_runs
    assert (False)

# Make sure libraries load
test_load_csv_lib
    import csv
```

Unit Testing Modularity

Application

author.py
author_test.py

article.py
article_test.py

comments.py
comments_test.py

Unit Testing Template

```
test_one_thing
    t = test_object
    x = process(t)
    answer = correct_answer
    assert (x == answer)
```

Unit Testing Exceptions

```
test_for_exception
    try
        t = test_object
        x = process(t)
        answer = correct_answer
        assert (False)
    catch
        pass
```

Pair Programming Guidelines

- ✧ Work in Pairs (1 keyboard + 2 brains)
- ✧ Switch for every iteration (micro-story)
- ✧ Test - Code - Refactor (Fail, Pass, Beautify)
- ✧ Typer - Talker
- ✧ Check your ego at the door —> Cooperate
- ✧ Save both product and test code
- ✧ Execute all tests for each micro-story
- ✧ Record a log of your time on each test
- ✧ Use the main script hack to run your code directly

Pair Programming Success

- ✧ Both people are fully engaged and focused on the problem.
- ✧ A major breakthrough happened
- ✧ Code works as desired
- ✧ Tests can be run 6 months from now
- ✧ Code is beautifully simple
- ✧ Either person is an expert

When do we do Pair Programming?

- ✧ Problematic code (nasty piece of work)
- ✧ Refactoring needed
- ✧ Tests needed
- ✧ Complex problem
- ✧ Cross-training
- ✧ Critical need for reliability

Author CRUD - Functions

- def add_author (name, email, password):
- def list_authors ():
- def get_author (name):
- def edit_author (name, email):
- def delete_author (name):

Article CRUD - Functions

- def add_article (user, title, body):
- def list_articles (user, title):
- def get_article (user, title):
- def edit_article (user, title, body):
- def delete_article (user, title):

Development Exercise

✧ Development loop

- Edit
- Test
- Integrate

✧ Create Author and Article

- Create
- Read
- Update
- Delete

Test-Driven Development

- ✧ Each feature
  - Select a feature
  - Write a failing test
  - Write just enough code to pass test
  - Refactor until beautiful

- ✧ Development travels at the speed of test

Version Control

- ✧ Create a folder in the Exercise Results with your BearID
- ✧ Convert your plan into Markdown
    - Project Plan
    - Technology Plan
    - Design Plan
    - Development Plan

Power of Wishful Thinking

- ✧ Top-down design
- ✧ Bottom-up construction
- ✧ Middle-out testing