Software Testing Methodology

Lecture 10

Gregory S. DeLozier, Ph.D.

A Brief Refresher

- So where were we?

Reasons For Failure

- Wrong requirements
- Defective code
- Defective tools

- Problem in execution environment
  - This probably is the most common reason for outage
  - Really expensive
  - Sophisticate software architectures are making it worse

Reasons for Environmental Failure

- Missing dependencies
  - Libraries
  - Tools
  - Volumes
  - Ports
  - Connections
  - Credentials

Reasons for Environmental Failure

- Lack of capacity
  - Memory
  - Disk space
  - CPU speed
  - Network Bandwidth
  - Parallel capacity

Reasons for Environmental Failure

- Changes in the environment
  - Additional activity
  - Capacity consumption
  - Connectivity issues
  - Patches
  - Interference – side effects from other work
  - Malware
  - Credentialing issues
  - Power failure

How does this relate to testing?

- Testing the environmental requirements
- Repeating that test on the target environments
- Periodically verifying the target environment
- Environmental assessment in case of failure

Programmatic Solutions

- Remote control of machines
  - Write function in, say, python
  - Part of the function is executed remotely

- These allow you to program remote machines
  - Functions to set things up – configuration
  - Functions to return values – testing

- Paramiko – remote control
- Fabric – convenient configuration
- Yarn – alternative for Python 3.x

Paramiko

- www.paramiko.org
- Implements SSH for remote login
- Very low-level
- http://jessenoller.com/blog/2009/02/05/ssh-programming-with-paramiko-completely-different

Paramiko Example

- import paramiko
- ssh = paramiko.SSHClient()
- ssh.connect('127.0.0.1', username='jesse', password='lol')

- ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy()) <- add

- stdin, stdout, stderr = ssh.exec_command("uptime")
- type(stdin)
- stdout.readlines()

  ['13:35 up 11 days, 3:13, 4 users, load averages: 0.14 0.18 0.16\n']

More Paramiko

```
ssh.connect('127.0.0.1', username='jesse',
    password='lol')
stdin, stdout, stderr = ssh.exec_command(
    "sudo dmesg")
stdin.write('lol\n')
stdin.flush()
data = stdout.read.splitlines()
for line in data:
    if line.split(':')[0] == 'AirPort':
        print line
```

Some Notes

- Paramiko is a challenging installation on Windows
- You can install Vagrant and use it there
- You can use it on PythonAnywhere (already installed)
- You can use it on Codio (easy to install)
- You can use it from any Ubuntu/Linux box

- If you really want to install it on windows, I can try to help.

Fabric – a Python 2.7 remote solution

```
from fabric.api import run

def host_type():
    run('uname -s')
```

If you save the above as `fabfile.py` (the default module that `fab` loads), you can run the tasks defined in it on one or more servers, like so:

```
$ fab -H localhost,linuxbox host_type
[localhost] run: uname -s
[localhost] out: Darwin
[linuxbox] run: uname -s
[linuxbox] out: Linux

Done.
Disconnecting from localhost... done.
Disconnecting from linuxbox... done.
```

Fabric Issues

- http://www.fabfile.org/
- Only Python 2.7 at this time
- Author is not porting
- Ports are available

Yarn – a Fabric Replacement

- https://github.com/Python-Yarn/Yarn
- Written in Python 3
- Uses Paramiko
- Very easy to use :

```
from yarn.api import env, cd, run


env.host_string = '192.168.1.2'
env.user = 'yarn'
env.password = 'yarn_is_aw3some!'

with cd("/usr/bin"):
    print(run("ls -al pytho*"))
```

Yarn Demo

- Demo Time

So… About Yarn

- Turns out that Yarn code is broken for use with passwords.
- Oops. ☺
- I wrote a simple replacement module – remote_api.py
  - Uses paramiko
  - Implements env/cd/run commands like Yarn & Fabric
- Someday Fabric will be ported.

Test Driven Infrastructure

- Design a test to verify some aspect of infrastructure
- Watch the test fail
- Fix the problem
  - Probably a dependency
  - Ideally, with automation
- Run the test again

Aside: the “private.py” file

- Use a private credentials file that doesn’t check in

```
user=”myusername"
password=”MyPa$$w0rd"
```

- Add “private.py” to .gitignore
- Then use this to get user and password

```
from private import user, password
```

Basic Example of a Remote Query

```
from remote_api import env,cd,run
from private import user, password

env.host_string = 'ssh.pythonanywhere.com’

env.user = user
env.password = password


with cd("~"):
    print(run("ls -l"))
```

Basic Plan

- Write Unit Tests to Verify Requirements
  - These will eventually become regression tests for the environment
- Write Remote Methods to Set Up Environment
  - These will make the tests pass
  - These will become the basis for automated deployment

Requirements for RemoteAPI

- Controller machine
  - Python3
  - Paramiko
  - remote_api.py
  - Access to target machine
- Target machine
  - SSH login capability
  - Bash shell
  - Credentials for login account (password or possibly key file)

Possibilities for Homework Access

- You can use Codio -> PA (paid, since Codio and PA ssh cost money)
- You can use Codio -> Codio (paid since Codio costs money)
- You can use your machine to ssh into KSU computer (free-ish ☺)
- You can use Vagrant
- You can use your machine to SSH into a raspberry pi
- …etc

- (In the future, I will be requiring a paid cloud account...)

Demo Time!

- All of this either is already in the repo or will be put there
- (Remind me to put it there!)

Mission:

1. Verify that the server is available
2. Set up the requirements for the Django application
3. Verify that the Django (superlist) application is installed
4. Verify that the Django application is running

Requirements Quick Reference

- We need a server
- git
- git config –global user.username, etc.
- Python3
- pip3 (pip3 install --upgrade pip)
- pip3 install django
- (pip3 install -- upgrade selenium)

Getting the Source

- Clone the repository or get the files
  - $ git clone https://github.com/hjwp/book-example.git
    - (then use branch command, or…)
  - $ wget https://github.com/hjwp/book-example/archive/chapter_07.zip

- Unzip the zip file
  - $ unzip *.zip
  - $ cd book*

Preparing the Application

- …In the source directory
  - $ cd book-example-chapter_07
- ...run the data migrations
  - “python3 manage.py migrate”
- …start the application
  - “python3 manage.py runserver”
  - This will terminate with logout, of course.
  - You can sleep and while sleeping run your webdriver tests

What Are The Limits?

- Obtaining servers from the cloud and verifying
  - Linode, DigitalOcean, Amazon, and Azure all have APIs
  - All are accessible from Python
- Initial configuration
  - Start with root account
  - Possible to change to user account for later setup
- Package installation
  - Remote operation of Git, etc.
- Application configuration
  - External configuration
  - Internal work via WebDriver

Does TDD work here?

- I claim that it does
- I claim that simple tools are better
  - (until they become inadequate)
- I claim that reliable setup leads to reliable environments
- I claim that this approach creates
  - reusable deployment assets
  - Useful regression tests for the environment and application
- Conclusion
  - This definitely works
  - It’s malpractice _not_ to use these kinds of tools.

Homework – last week revisited

- Get a remote computer running
  - Vagrant or cloud computer
- Execute some ~~yarn~~ ***remote_api*** commands against it
- Use python unittest framework to test some setup parameters
  - Is <?> installed?
  - Is <?> directory created?
  - Does this directory have the right permissions?
  - …etc
- As always, send me something showing that you did this.

Homework – Last two weeks

- Use the remote API methods to test an application in all phases:
  - Test the environment, packages, etc.
  - Test that the application is installed correctly.
  - Test that the application is running.
  - Test that the application is working correctly, using the tests previously written for the application.
- You have to have a running application to do this. If time allows, automate the installation of the application.
- For what it's worth, at the end of this lecture, the application was up and running and we could have opened it in a web browser.
- I'm having a difficult time finding a way to have remote_api start the django app and leave it running. It's not something we normally do with the django development server. You may either start the server manually, or wait for me to post something, or you can try out nginx per the book instructions.

Homework Rules

- On the subject line:
  - “Testing” or “STM”
  - “HWn” where n is a number
  - A phrase describing the work
    - (NYPL, Calculator, etc)
- In the body of the email:
  - Your Kent State email address as text (e.g. gdelozie@kent.edu)
  - Email text – whatever you want to say
  - PDFs
  - Images (jpg, png)
  - ***NOTHING ELSE***
- ***All homework is due before the class session in finals week***

Things we will cover in “lightning talks”

- We will cover the following additional chapters:
  - 11 – Forms
  - 13 – Javascript
  - 15,16 – Mocking
  - 17 – Logging
  - 18 – Outside-In
  - 19 – Test Isolation
  - 22 – Test Architecture and Topics

Regarding Software Engineering (Spring 16)

- We will be using this book for some topics in SWE

  - Continuous Integration
  - Deployment preparation
  - Deployment management and DevOps
  - Environment validation and testing
  - Authentication and Security
  - Database migration
  - A few other things here and there. ☺