example_project Documentation

Release 0.1.0

Roie R. Black

September 27, 2017

Contents

Contents:

CHAPTER 1

Development Services

When you set out to build a significant Python project, you could work alone until your project is ready for release into the "wild". You could do that, but you would miss out on a huge benefit you get by making things public long before it is really ready to go.

There are a bunch of free services you can use to make sure your development is going smoothly as you work through the design. Tapping into these services can help you spot problems and fix them early in the process. Here are a few of the free services you can use:

GitHub

This is a big one. Employers are telling us they want to see your work on GitHub, regardless of the programming language you choose to use for your project. GitHub is free for *open-source* projects, or you can pay a fee to keep your projects private if your choose. Alternatively, I set up a private server running a clone of GitHub, called GitLab, for use by my students as part of their work in my classes. GitLab provides services similar to GitHub from their project website. You can also install GitLab on a Linux server and host it in your home office using a *Dynamic DNS* service. This will allow you to reach your server from anywhere you can get on the Internet. GitLab also offers free hosting for your project if you do not want to run your own server.

To get your project on GitHub, you sign up for a free account using a username and password. Your username becomes part of the URL your project will get. For example, I set up an example project named `example_project` for this lecture. GitHub set up a URL for the project at `https://github.com/rblack42/example_project.git`. In this URL, my username is `rblack42`, and the project name is ``example_project`. GitHub_ added the ``.git` part.

As you create the project, you have a chance to set up a basic `README` file, a `.gitignore` file that is designed for the programming language you intend to use, and a basic license file. (I use the `New BSD` license for my projects. By adding these few files, your project will be ready to clone onto your development system:

```
$ git clone https://rblack42/example_project.git``
$ cd example_project/
```

You will end up with a project directory named `example_project` on your system already set up so Git can manage it.

virtualenv

If you work on several Python projects on the same development machine, eventually you will run into dependency issues. You have one package that needs another one to function, and these packages end up needing conflicting

versions of each other. Ian Bicking solved this problem by creating the virtualenv package which sets up an isolated Python environment where you can install only those packages (with correct versions) needed by your project. Furthermore, you can set up a `requirements.txt` file and list all the packages and versions you need, and let `pip` install everything for you. Most serious Python developers use these tools as part of their normal work flow!

As an example, on my Mac I installed virtualenv using this command:

```
..  code-block:: text

    $ pip install virtualenv
```

Next, I added an alias to my `.bash_profile` that looks like this:

```
alias workon='source _venv/bin/activate'
```

Finally, in my newly cloned project directory, I do this:

```
~/_projects/example_project$ virtualenv _venv
New python executable in _venv/bin/python2.7
Also creating executable in _venv/bin/python
Installing setuptools, pip...done.
```

When I want to work on the project, I do this:

```
~/_projects/example_project$ workon
(_venv)~/_projects/example_project
```

Notice that the prompt has changed to remind you that you are working in a "virtual environment". There is a command to deactivate this environment, but I usually just close the window.

Note: I have a habit of naming some directories with a leading underscore. That makes these names appear at the top of any directory listing I see, so I can get to it quickly as I navigate around in the file explorer tools.

PyTest

A nice Python package for managing tests is PyTest. Now that we are working in a virtualenv, we can install this package easily using `pip`. To demonstrate how the requirements.txt file works, here is the firat line I add to this file in the new project:

```
pytest==2.6.4
```

Then you install it by doing this:

```
$ pip install -r requirements.txt
```

Note: If you leave off the version part ('==2.6.4'), you get the most recent version. After doing that, you should update your *requirements.txt* file to reflect the version you actually used in your project. That way, if the project evolves, you can still install the correct version. You can update your project to use the new version as needed at a later time.

Something to Test

Now that PyTest has been installed, let's give it something to test. Normally we keep all tests in a separate directory. You may or may not decide to release the tests with your project. I feel that releasing the tests is a good idea, since you

can find out if there are any problems in unexpected environments by letting your users test things and report back if any issues pop up!

Here is the first test file, named `test_dummy.py`. Create a new `tests` subdirectory for this file:

```
def inc(x):
    return x+1

def test_inc():
    assert inc(3) == 4
```

Before we run this test, we need to tell PyTest not to look into our virtualenv subdirectory, or else it will try to run tests we are not really interested in running. To do this, create a `pytest.ini` file in the project root directory with these lines:

```
[pytest]
norecursedirs = _venv
```

With this test and control file in place, we can run the test:

```
$ py.test
============================== test session starts ===============================
platform darwin -- Python 2.7.8 -- py-1.4.26 -- pytest-2.6.4
collected 1 items

tests/test_dummy.py .

============================ 1 passed in 0.03 seconds ============================
```

Great! Our project is on track with a real (silly) test that works, and we have not even started writing any real project code yet! Time to let the world know how our project is doing!

TravisCI

TravisCI is a nice free service that works with GitHub to test your project code. You can sign up for this service using your GitHub credentials, making the process pretty easy! TravisCI will download a list of your public projects from GitHub and you can select the ones you want to test with this service by clicking on a menu item.

You need to add a new file to the root of your project folder, and commit everything in your project, then push your changes up to GitHub.

Here is a starter `.travis.yml` file for this project:

```
language: python
python:
  - "2.7"
  - "3.4"

# command to install dependencies
install:
  - pip install -r requirements.txt

# command to run tests
script:
  - "py.test ."

after_success:
  - coveralls
```

Before we check all of this in to GitHub, we need to see what Git thinks about everything:

```
$ git status

On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .travis.yml
    _venv/
    pytest.ini
    requirements.txt
    tests/

nothing added to commit but untracked files present (use "git add" to track)
```

We do not want the virtualenv directory to end up on GitHub so we edit the `.gitignore` file to add these lines at the end:

```
# virtualenv
_venv
```

Rerun `git status` to confirm that Git is no longer looking at that directory, then commit and push your changes:

```
$ git add .
$ git commit -m "initial project setup"
$ git push origin master
```

After this push, you can navigate to your TravisCI account page and watch the action. Travis will clone your project into a new virtualenv and run your tests. Depending on the load on their servers, this may take a few minutes.

Here is the last part of what I saw on my build run:

```
git.checkout
0.05s$ git clone --depth=50 --branch=master git://github.com/rblack42/example_project.git rblack42/ex
Cloning into 'rblack42/example_project'...
remote: Counting objects: 16, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 16 (delta 3), reused 12 (delta 3), pack-reused 0
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (3/3), done.
Checking connectivity... done.
$ cd rblack42/example_project
$ git checkout -qf 8b153311fd5bcdc054dd67e000a0f2e5b849a056
0.01s$ source ~/virtualenv/python3.4/bin/activate
$ python --version
Python 3.4.2
$ pip --version
pip 6.0.7 from /home/travis/virtualenv/python3.4.2/lib/python3.4/site-packages (python 3.4)
install
0.52s$ pip install -r requirements.txt
You are using pip version 6.0.7, however version 6.0.8 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Requirement already satisfied (use --upgrade to upgrade): pytest==2.6.4 in /home/travis/virtualenv/py
Requirement already satisfied (use --upgrade to upgrade): py>=1.4.25 in /home/travis/virtualenv/pytho
0.34s$ py.test
============================= test session starts ==============================
platform linux -- Python 3.4.2 -- py-1.4.26 -- pytest-2.6.4
collected 1 items
```

```
tests/test_dummy.py .

========================== 1 passed in 0.05 seconds ==========================

The command "py.test" exited with 0.
Done. Your build exited with 0.
```

Once you set things up, every time you push changes to your project on GitHub TravisCI gets notified and it checks our a copy of your code into a brand new virtual server, then runs your tests to see that everything is working. If so, you can post a "badge" on your project `README` page so folks can see that the project is in good shape. On the other hand, if your project fails any of the tests you have set up, the badge will show that as well, so potential users of your code will shy away, or might help you fix things if this is a cool project! *Open-source* can be fun!

To see your new badge, add these lines to the `README` file and get it up to GitHub (which will trigger a new build, by the way!). I like to use reStructuredText markup in my `README` file, and GitHub formats this nicely for the web browser.

```
Example Project
###############

:author:    Roie R. Black
:version:   0.1.0

..  image:: https://travis-ci.org/rblack42/example_project.svg?branch=master.

This project is a demonstration of many ``continuous integration`` services
for Python developers.
```

And, here is my newly issued badge:

README.rst

Example Project

Author:	Roie R. Black
Version:	0.1.0

build passing

This is an example project demonstrating the setup of a Python project that uses freely available `continuous integration` services.

That green badge is what developers stay up late at night to see. If the build fails, the badge is red and says "failing". Not a good situation to be in when you stop working!

AppVeyor

Another nice free service that can test your application is AppVeyor. This project is unique in that it will deploy and test your project on a Windows server. If you expect your users to want to run your project on Windows systems, this

service is a must!

Once again, you can use your GitHub credentials to set up an account here.

You need to add a configuration file in the project root, and this is another YAML file. I had some issues building this file, since it is very picky about spaces use din indenting. Fortunately, there is a appveyor.yml validation page on the AppVeyor site that will check your file before you push it to GitHub.

Here is the basic file I set up:

```
version: '{build}'
environment:
  matrix:
    - PYTHON: "C:/Python27"
    - PYTHON: "C:/Python34"

init:
  - git config --global core.autocrlf input

install:
  - ps: (new-object net.webclient).DownloadFile('https://bootstrap.pypa.io/get-pip.py', 'C:/get-pip.p
  - "%PYTHON%/python.exe C:/get-pip.py"
  - "%PYTHON%/Scripts/pip.exe install --upgrade setuptools"
  - "%PYTHON%/Scripts/pip.exe install -r requirements.txt"

build: False

test_script:
  - "%PYTHON%/Scripts/py.test"
```

Now, when we push changes to the project to GitHub, both TravisCI and AppVeyor will run builds to verify that the project works properly on both Linux and Windows. (Now, I am wondering if there is a similar service for Macs!)

AppVeyor also creates a badge for your `README` file. To get the URL for your badge, check on the `settings` menu for the project. I added the URL to my `README` file, and this is the result:

Hey! We are making progress!

CoverAlls

WHen testing code, one of the metrics we should monitor is the percentage of the code in the project that has been tested. This is not a complete measure of the quality of the code, but there is an old saying:

> If it has not been tested, it does not work!

So, we want to make sure all lines of cod ein the project have been exercised during some test.

There is a nice tool for Python projects that can do this check: coveralls. We install this package by adding another line to our `requirements.txt` file:

```
coveralls==
```

There is another free service Coveralls.io, that will create a nice report on your coverage statistics.

Landscape

- http://landscape.io/

This is a free service for *open-source* projects.

PyPi

Version

Downloads

Wheel

Supported Versions

Scrutinizer

ReadTheDocs

Every project needs documentation, and the best projects have excellent user level, and developer level documentation. Much of the documentation is produced using python Sphinx which is the standard tool used by the Python project itself.

ReadTheDocs is a web site that will post your documentation for the world to see. All you need to do is generate then using Sphinx as part of your project on GitHub, then set up ReadTheDocs to pull down new versions every time you commit changes to your code. (You do update the documentation as well, right?)

Sign up for a free account on ReadTheDocs and use their web menu to set up the project. Make sure you add SPhinx to your `requirememts.txt` file:

```
sphinx==1.2.3
```

Now, each time you commit changes, your documentation will be updated.

Tox

Tox is a command line tool that checks to see if your project works properly with multiple versions of Python. It will create a *virtual environment* for designated versions of Python, install your project in those environments, then run test to make sure everything works. You can use Tox as a standalone tool on your workstation, or have TravisCI automatically do all of these tests for you.

To use Tox in your project, you add one file, `tox.ini` to the root of your project. Then, you run the `tox` command in that project directory. Here is a starter `tox.ini` file:

```
[tox]
envlist = py27, py34

[testenv]
deps=pytest
commands=py.test
```

This setup assumes that the project is set up to run tests using PyTest, and the script will run the tests using Python 2.7 and Python 3.4.

Putting all this Together

To see how to set all of this up, let's create a dummy project with all of these services at work to check things. Stand back, this will involve a bit of work!

Step1: Create the project directory

Step2: Get the project on GitHub

Step3: Add some sample code

Step5: Add a few tests

Step6: Set Up TravisCI

Step7: Set up your Documentation

Step8: Add in ReadTheDocs

Step8: Add in

CHAPTER 2

Glossary

Acceptance Tests, System Tests, User Tests Testing of the complete application to confirm that it meets the project specification

Accessor, Mutator Methods that can access a private attribute. These methods control access to the attributes and can enforce value validation to make sure the attribute is properly constrained.

API, Application Programming Interface A collection of methods that provide the public interface to a subsystem used in your application.

Assemble, Assembler, Assemblers When we process a program written in *Assembly Language*, we use a tool called an *assembler* to convert the program into *machine language* the processor actually understands. We *compile* our programs.

Assembly Language A human readable form of *machine language*. An *assembler* converts code written in this language into *machine language*. These languages are processor specific.

Avatar A graphical representation of a user. Often an image, but it can be a cartoon-like character as well. To make the web more personal (and less anonymous), sites like Gravatar help users show their chosen avatar on many web sites. I consider this usage part of setting up a *professional image*.

Baby Step, Baby Steps A software development process that focuses on making very small changes to your code, followed by a quick test using the compiler to make sure you can run your code, and a run that generates output you can inspect to see how your changes are working. The key to this is doing this sequence often.

Blasting Code, Blast Code An *old school* method of software development. This method involves long sessions of writing code with few attempts to test the code. The result is often long sessions fixing typing mistakes, followed by long sessions with a debugger trying to figure out why things do not work.

Branch, Branches A fork in the development road where a new feature is developed separate from the main line of development. Eventually this new line of development will be *merged* back into the main line.

Camel Case A naming convention where the name is made up of multiple words, each with the first letter capitalized, and spaces removed. For example: *CamelCase*.

CAS, Content Addressable File System A scheme used by Git to store documents based on a hash code generated from a file's contents. The hash code is used to form a directory name where the file is stored.

Change Script, Change Scripts This is a chunk of code that can be used by an appropriate tool to convert one product into another one. These scripts are the heart of *source code control systems*. The Unix tool **diff** is used to create such scripts.

Clone Create a copy of files living on a remote server for your local use. This creates a *working copy* of the project. You may (or may not) have rights to make changes and *push* them back up. It is common to *clone* project hosted on services like GitHub so you can use the code or study it. You can update your local copy at any time to keep up with changes in the project.

Code of Conduct Every profession has a defined set of rules they expect members of that profession to follow. Most of the time, these rules are common sense, but they are defined to make sure every member understands what the profession seeks to present itself to the public. You are expected to know these rules and follow them when you join the profession. Look into each profession's primary organization for guidance. For computer folks, this is probably the Association for Computing Machinery.

Command A series of space-separated text items handed to the operating system. The first of these items is the name of some program (which may be internal to the operating system) you want to run. The rest of the items are called *parameters* which are processed by the program and control exactly what that program does.

Command Line A simple text line on your screen where you can type in a *command* to the operating system.

Comment, Comments Short chunks of text added to program code that seeks to explain when is going on. Far too often, this text just repeats what the code actually says, making them useless. Comments should explain the "why" of a block of code.

Commit, Commits *push* to a remote server.

Commit, Commits The process of generating a marker that records the state of a project at that moment in time. *SCCS* systems generate a *change script* that can be used to restore your project to the exact form is has at this moment, even as the project continues on in its development.

Compile, Compiler, Compilers When we process a program written in a high-level programming language, the tool we use is called a *compiler*. That tool converts our program into the *machine language* the processor actually understands.. We say we *compile* our programs.

Conflict, Conflicts A situation where two different versions of a single project asset exist. The differences must be resolved before the asset can continue to exist in the combined project.

Content Addressable File System A system that stores objects with names based on the contents of those objects. Used by Git internally to store version information.

Context Free A programming language must be defined in such a way that figuring out what should come next (*tokens*) can be figured out without studying the context in which the *token* appears.

Context Menu On most systems, when you `right-click` on something, you bring up a menu that is sensitive to the context of the thing you are pointing to. That menu lets you perform tasks common to that thing!

Continuous Integration A testing technique where a project is automatically tested on a number of build systems to verify that the project is running properly. By testing automatically, developers can spot major problems that cause errors on specific platforms quickly.

Cookie A small file containing information needed to reconnect you to a web server automatically. Cookies often store log in credentials. It is vital that you never allow your personal cookies to be saved on a public machine. Doing so gives others complete access to your accounts. Parameter Parameters A block of text (no spaces allowed, unless the block is enclosed in quotes) that is passed to a program or a sub-program. If passed to a program and there are more than one parameter needed, they are usually separated by spaces. If passed to a sub-program, they are usually separated by commas.

Current This is the flow of electrons through a circuit. Moving electrons generate heat and electromagnetic waves as a by-product of this motion.

Data Alignment Modern processors work best when data is aligned so that the data item fits in a natural data chunk the processor will fetch. Misalignment problems occur when a single data item requires two memory accesses to access.

DDNS, DynDNS, Dynamic DNS A service that updates *DNS* tables used by Internet services to map domain names to addresses. These services are helpful if your network address changes often, as it might if you are working from home and want to set up a server.

Debugger A software development tool that is used to step a program one line at a time, then allows the developer to inspect the internal state of the machine as the program runs. They are a vital, and often under-used tool in programming.

Decode After the processor has *fetched* part of an instruction, it decodes that part to determine what additional information it needs to do its work. If it needs more data from memory, that data will be read from memory into internal registers. As part of this step, the complete size of the instruction is calculated, and the *IP* register is updated to point to the next instruction in memory.

Diff, Difference A tool can read two files and show the differences between them. It can also create a *patch* file that can be used to convert one of them into the other.

Directory, Directory Tree The *operating system* on most computers, today, uses tree-like structure to store files. The top of this tree, called the root of the tree, is a directory (or folder as Microsoft want us to call it). The name of this directory is either "/" on a Linux/Mac system, or "" on a Windows system.

A Directory is a container that can hold either files (objects that contain only data, such as text, images, of other binary data) or other directory objects (called subdirectories). Subdirectories have names that follow the rules for the *operating system*. Any directory or subdirectory can hold files of further subdirectories, giving the entire file system a (upside down) tree-like structure.

DNS, Domain Name Service A phone book like service that maps domain names, like `www.pylit.org`, to an IP address.

Domain Name Servers on the Internet are all part of some registered domain, which has a name associated with it. I own *pylit.org* for example. The last part is used to brand the kind of domain I am part of, in this case a non-profit organization. The first part is often chosen to identify the company involved. You can add a machine name to the front of this name. Often *www* is tacked on to identify a web server where the companies website is found.

DotFiles Many programs keep settings in a hidden file in the user's *home directory*. Exactly what these file hold depends on the program. It is common to manage these files in some way. I keep mine on my GitHub account at https://github.com/rblack42/dotfiles.

Drag and Drop A modern method of constructing programs, or executing commands by dragging an icon on the screen and dropping it onto another place. Learning how all of this works can make you highly productive!

EBNF, Extended Backus-Naur Form A notation used to define the *syntax* of a programming language. The notation is designed to be *context free* meaning there are no places where figuring out what comes next in a program depends on the context of the program. Usually, the rules can tell what is coming next by simple looking at the next *token* in the program

Environment Variable, Environment Variables Named strings managed by the command line processor. These strings are available to programs and are commonly used to set up data for a variety of purposes.

Execute, Executes, Executable, Executable File In spite of the negative connotations of the term, we say a computer *executes* a program. The program that can be run is called *executable*.

Execute, Execution We call the act of actually processing something in a computer *executing* that something. The something can be a single *Machine Instruction*, or a complete program. We are a brutal race of beings, aren't we?

Fetch, Fetched Using a special register called an *Instruction Pointer* (or *IP*, the processor reaches into the system memory at the address designated by the *IP* register and fetches a defined number of bytes which form all or part of the next *Machine Instruction* to be processed.

Finite State Machine A system whose operation is defined as a set of *states* and transitions from one *state* to the next.

Flow Chart A form of diagram showing the logic of a program. These diagrams have been around for almost as long as computers have been available. They are a great way (but not the only way) to visualize how your program will "flow" and think about what will happen when you run the program.

Gravatar A service that will provide your image when you log into a number of web sites. That image can be viewed by others to make sure they recognize the you they are communicating with. I consider using such a service part of setting up a *professional image*.

GUI, Graphical User Interface A user interface were the mouse and windows are used to control applications.

Hash, Hash Key A string of characters generated using a has function that is applied to all of the content of some object. These strings are often used to detect if the object has been altered in any way.

Hidden File Most operating systems "hide" file names that start with a dot ".". These normally contain configuration information that should not be modified unless you know what you are doing You need to use a special command to display these files when generating a directory list.

High-Level, High-Level Language, High-Level Languages A language designed to be compiled into a *low-level* form like *machine language* is called a *high-level language*. Such languages are designed to be *machine independent*. C++ is a *high-level language*.

High-Level Language Most programming languages are designed to help humans instruct a machine in how to solve some problem. These languages do not care what processor they will run on, and are called *machine independent*. You use a *compiler* designed to convert your high-level code into *machine language* for a particular processor. Different *compilers* support different machines.

Home Directory Most operating systems create a directory you are to use for all of your files on that system. This directory is tied to the user account you log into when you access that system. On Windows, this is the directory where your "My Documents" folder is found, usually a place like `C:\Users\username`. On Linux systems the directory will be in `/home/username`. On Macs, it will be in `/Users/username`.

IDE, Integrated Development Environment, Integrated Development Environments A collection of common programmer's tools integrated into a single application with features that can greatly speed up program development. Unfortunately, typical IDE systems are complex and may not support all the languages you use, or be available on all the platforms you use.

Instruction, Instructions, Machine Instruction, Machine Instructions A single instruction that a given processor can *execute* is called a *machine instruction*, or just an *instruction*. A set of *Machine Instructions* make up a *Machine Language*

Instruction Pointer Processors use a special *register* to keep track of the address in memory where the next *machine instruction* to be processed can be found. This *register* is called the *instruction pointer*, or just the *IP* register.

Integration Tests Tests of a set of units to confirm that they work together properly.

Interrupt, Interrupts, Interrupt Handler Processors normally run programs using a simple four step process called "Fetch-Decode-Execute-Store". When the machine is at the point where it is about to fetch another *instruction*, a signal can stop the action, and cause the machine to jump to a block of code that handles the signal. The signal is called an interrupt, since it interrupts the normal flow of a program. The code that deals with the signal is called an *Interrupt Handler*. Hopefully, *interrupts* are short enough that the interrupted program does not notice the delay!

IP Internet Protocol. This term refers to the "dotted-quad" number assigned to your machine when it is connected to a standard network. You can determine your "IP number" by running *ipconfig* on a PC, or *ifconfig* on either a Mac or Linux system.

Latency The delay between when an action is started and when it completes. In computing, this is usually measured in clock cycles.

Link, Linking, Linker A phase in transforming your program into a final *executable* file where one or more *object files* are combined with system libraries' to build a final *executable* file. The tool that does this work is called a *linker*.

Literate Programming I form of documentation where program fragments are included in a literate document designed to explain how a program is developed. Those fragments are extracted from the documentation and

combined into a normal form that can be processed by programming tools. This is the exact opposite of traditional documentation where the code is littered with *comments* that try to accomplish the same thing. These *comments* are often neglected and become meaningless.

The creator of this concept is Donald Knuth. If you claim to be a software developer, you owe it to yourself to read his work.

Low-Level, Low-Level Language, Low-Level Languages A language that is very primitive, often tied to a specific processor. *Assembly language* and *Machine language* are *low-level languages*.

Machine Dependent, Machine Independent Languages are either specific to a processor (*low-level languages*) or independent of any processor (*high-level languages*).

Machine Language, Machine Code The *low-level* binary language a particular processor understands. *Machine Language* is a set of very simple instructions that the processor can *execute*. These languages are designed by the processor manufacturer, and are unique to that processor (or processor family). Usually that manufacturer designs an Assembly Language to go along with their *machine language*, but you are not required to use that language. As long as your *assembler* generates correct *machine language*, your code can run on that processor.

Master Branch The primary development branch in a project managed with a *SCCS*.

Master Copy, Master Server Development teams usually set up a server for use by the team. A copy of the project files is kept on this server and all team members make sure to synchronize their work with this server. We *push* your changes to the server, and *pull* changes down from the server, so we can see work done by other members of the team. Command Line A simple text line on your screen where you can type in a *command* to the operating system.

Merge, Merged Combining two distinct development procedures into one. This may result in *conflicts* that must be resolved.

Mock Object, Mock Objects A program component that implements the interface to a complex subsystem sufficiently well to enable testing of code that uses the real subsystem.

NTP, Network Time Protocol, Network Time Synchronization Servers on the Internet with access to very accurate (usually based on atomic vibrations) time signals, keep track of the current time and can report that time to your system using a simple protocol. Most machines connected to the Internet can be set up to synchronize their view of the current time with these servers. (The software can even account for the time it takes for those signals to reach your machine!)

Object file, Object files A file containing *machine code* for a particular machine, but missing some references needed to build an *executable file*. Several *object files* are combined, resolving the missing references to build a program. Usually, some of those unresolved references are resolved by searching a *system library* that accompanies the transformation tool (*compiler*) or the *operating system*. Only when there are no *unresolved references* can your program actually *execute*.

Old School The place where old methods were learned. These methods are any methods not accepted as modern by current workers.

OOP, Object Oriented Programming A design technique where the fundamental program components model objects from the real world the application is to serve.

OPC, Other People's Code You should study code written by other folks, especially those who seem to do the job well. Eventually, you will learn how to write well respected code and become the author of what others read. Just make sure you give credit for anything you decide to incorporate into your projects, and respect the license that goes with the code.

Open-Source An open-source project is freely made available to the public in source code form. Such projects are usually protected by a license of some kind, designed to protect the rights of the author(s).

OS, Operating System All general purpose machines need a piece of software to manage the hardware of the system. Typically, these programs provide a user interface to make controlling the machine simple. No on buys a system

just for the *OS*. Instead, you pick an *OS* based on the applications you need to use to do real work (or play) on that system.

Over-Clock, Over-Clocked It is possible to program the clock on some systems so it runs faster than advertised by the manufacturer. You might get more speed out of your system by doing this, but you also might reach a point where the signals are not getting where they need to be in time, and the results are totally unpredictable.

Parameter, Parameters Values passed to a sub-program for it to use in it's work.

Problem Statement Every programming project should start with a problem statement that you analyze to see exactly what you are supposed to create. Your job is to create a solution to some problem. You may need to go to the originator of the statement to clarify things, and you should do this rather than guess at what is really wanted.

Professional Image You have an image on the Web. Like it or not, this image follows you everywhere, even into the first job interview. Are you proud of that image? May folks post the most outrageous junk on their FaceBook pages, thinking no one but their "Friends" look at it. Not so! Unfortunately, your potential employer might check you out and not like what they see. Your choice. I recommend making sure you look like someone others want to employ, or at least someone who does more with their lives than just keep everyone informed about every aspect of your life on FaceBook! YMMV!

Provisioning Tool, Provisioning Tools These programs help system administrators install and manage systems by automating the previously tedious process they used to perform. There are several tools in this category: Puppet, Chef, and Ansible are all popular *open-source* today.

Push, Pull When working with an *SCCS master server*, we *push* your changes to the server, and *pull* changes down from the server, so we can see work done by other members of the team.

Real World A fictitious place everyone claims exists. In this world things work perfectly (or at least better than in your present world!)

Refactor Modify a programs code to improve its quality without changing how it works. This is a clean-up step in development designed to keep ugly code out of a project, and use best-practices in how code is presented.

Refactoring Rewriting your code to make it simpler, and easier to understand without changing how it works. Your *regression tests* will tell you if you are doing this right!

Register A place inside the processor where data can be stored. There will be a number of such places in each processor Instead of addresses, these locations will have names. We refer to those names in assembly language programming.

Regression Test, Regression Tests Testing to make sure a project is moving forward, not backward.

Repo, Repository, Repositories A file system that serves as a database for a source-code control system like Subversion or Git.

reStructuredText A simple markup language designed to make documentation readable with no processing. Processing tools like SPhinx can turn documents written in this markup into HTML or PDF output that looks very nice!

Revision, Revisions Another term for *version*. A *revision* captures the state of a collection of assets (files) at a particular moment in time. :term:Source Code Control Systems' track the changes from one *revision* to another by creating *change scripts* that can convert the older version to the newer one.

SCCS, Source Code Control System, Source Code Control Systems Distributed Source Code Control Systems A tool used to track changes to all files involved in a development project. Most such tools use a central server. The distributed tools can function with no server as long as team members can communicate directly over the Internet. Typical tools are Subversion and Git

Script Programs designed to control computers are often called scripts. Most scripts are written using simple languages like *bash* on a Linux system or Python.

Semantic Analysis The process of taking a properly formed construct in a programming language and converting it into another form the has the same meaning as the original construct. Typically, a tool like a *compiler* transforms your high-level code into *machine language* the processor can actually run.

Semantics A properly written programming language construct causes the computer to do something. Exactly what that something is is called the semantics of that construct. You must understand the semantics in order to make sure you use that construct properly. This is called picking the right tool for the job!

Semantics This term refers to the meaning of a construct in a programming language. Only a construct that passes the *syntax analysis* phase of transformation has any meaning.

Shell, Command Prompt The "old fashioned" way of controlling a computer. On Windows systems, this involves opening a window where you can type in commands. On Mac and Linux systems, you open up a **Terminal** program.

Side Effects Functions usually operate as self contained blocks of code. They get information from the outside world through the parameter list, and produce results that are returned to the caller as return values. If the function modifies anything in the outside world as it performs its task, these are called side effects. Generally, these are bad things, and should be avoided.

SSD, Solid State Drive Most hard disks use a rotating metal disk coated with a magnetic material that records the zeros and ones. A modern (and a bit expensive) alternative is to use electronic circuits similar to flash memory, but much faster, to store the bits. Since these drives are much faster than traditional hard disk drives, some systems use them to store the operating system, so the machine boots fast. If you have enough money, an *SSD* might be the only drive you have.

State, States The exact configuration of a collection of items that make up some system. That set of items can grow or shrink over time, and individual items may be changed. At any moment in time, the entire collection is in some configuration.

The operation of many systems can be described exactly by a sequence of specific changes from one state to the next. Such a system is called a *Finite State Machine*.

Store, Retire, Retirement Once a single *Machine instruction* has been completed, any new data produced by that instruction needs to be written to a final storage location, clearing the processor to move on to the next instruction. We say we *store* that data, or *retire* the instruction.

Style Guide A document specifying the style to be used to projects in some organization. These guides lay out how program code should be presented, how variable names are formed, or how files should be organized to meet standards everyone in the organization will follow. Some of these guides can be extensive, others, very informal.

Syntax The formal rules that define how to write a construct in a particular programming language. These rules are simple and precise.

Syntax This is a term referring to exactly what a statement in a programming language looks like when written in a program.

Syntax Analysis, Syntax Analyzer The process of examining each construct in your program to make sure it is properly formed. Only when this is true can the analysis tool move on the *semantic analysis* to figure out what each construct means. The tool that checks the syntax is called a *syntax analyzer*, usually part of a *compiler*.

Syntax Diagram, Railroad Diagram A visual representation of a rule in *EBNF*. These rules define the *syntax* of a language.

System Clock The master timing device on all computers. The processor uses the clock to synchronize the execution of machine instructions.

System Library A file containing references needed by programs to connect to operating system of language specific functions.

System path Most operating systems search for files to execute by examining directories listed in a system variable called the `PATH`.

Tag, Tags A special marker used to identify particular *revisions* in a software development process.

TDD, Test Driven Development A software development process where any change to the code is preceded by the creation of a test that will demonstrate the operation of the code after the change is made. The development process becomes one of generating code that passes the test.

Test Driven Design A fictitious design technique where we practice *TDD* with no design in mind. This might be a bottom-up technique where interesting subassemblies get produced that might be useful in a larger project. Basically, this seems like a bad idea.

Texas Four-Step All computer processors use a four step process to do their work. The four steps are called *Fetch*, *Decode*, *Execute*, and *Store* (or *Retire*.

Token, Tokens In *syntax analysis*, your program is broken up into a number of *tokens*, a term describing one item that is "indestructible" in the language. For example the keyword "if" is a single token, so is an integer number, or a curly bracket, or the ">>" symbol used in C++.

Unit, Unit Test, Unit Tests A unit is a basic component of a program. Typically this is one function or a class. We test these units to verify that they work as we intend.

Unresolved Reference, Unresolved References When you try to link your program and there are references to items that cannot be found, we say these are *unresolved references*. The final *executable file* cannot be constructed in this case. System Library System Libraries Files containing code needed by a program to actually complete the transformation of a *high-level* language into *machine language*. In these libraries, we find code for input and output, code needed to interface with the *operating system*, etc.

URL, Uniform Resource Locator The specific location of a web "resource", which is normally a web page, but can be a file or anything that your browser can download to your system using the web protocol named *HTTP*.

VirtualHost A configuration used by Apache to manage web servers controlling multiple websites

VM, virtual machine, virtual machines A program that emulates a real machine accurately enough to run real programs for that machine.

VM, Virtual Machine A program running on a host computer that emulates a real machine well enough that operating systems and applications can run inside them. This effectively isolates the environment inside the *VM* from the host computer.

Voltage This is a measure of the "force" driving electricity through a circuit. Think of pressure in a water pipe.

VPS, Virtual Private Server A *virtual machine* set up on a remote server in such a way that a user can use the *VM* as a private serer. This arrangement is commonly used by companies to make more effective use of physical servers. Multiple *VPS* systems can run on one physical machine. Companies like Rackspace and Amazon offer access to such systems.

Working Copy, Working Copies When using an *SCCS* system, the copy maintained on a remote server is usually treated as a master copy. Any local copies are called *working copies*. When you leave any work session, you should try to make sure the remote and working copies are identical. This will prevent problems later!

WSGI, Web Server Gateway Interface A standard protocol used to host Python applications on web servers

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W