nodev tutorial Documentation

Release 0.2.0

Alessandro Amici

2016-07-22

Version 0.2.0

Date 2016-07-22

Warning: This documentation is work in progress and there will be areas that are lacking.

CHAPTER 1

Test-driven code search concepts

1.1 Motivation

"Have a look at this piece of code that I'm writing–I'm sure it has been written before. I wouldn't be surprised to find it verbatim somewhere on GitHub." - @kr1

Every piece of functionality in a software project requires code that lies somewhere in the wide reusability spectrum that goes form extremely custom and strongly tied to the specific implementation to completely generic and highly reusable.

On the *custom* side of the spectrum there is all the code that defines the features of the software and all the choices of its implementation. That one is code that need to be written.

On the other hand seasoned software developers are trained to spot pieces of functionality that lie far enough on the *generic* side of the range that with high probability are already implemented in a **librariy** or a **framework** and that are documented well enough to be discovered with a **keyword-based search**, e.g. on StackOverflow and Google.

In between the two extremes there is a huge gray area populated by pieces of functionality that are not *generic* enough to obviously deserve a place in a library, but are *common* enough that must have been already implemented by someone else for their software. This kind of code is doomed to be re-implemented again and again for the simple reason that **there is no way to search code by functionality**...

Or is it?

1.2 Test-driven code search

To address the limits of keyword-based search *test-driven code search* focuses on code behaviour and semantics instead.

The **search query** is a test function that is executed once for every candidate class or function available to the **search engine** and the **search result** is the list of candidates that pass the test.

Due to its nature the approach is better suited for discovering smaller functions with a generic signature.

pytest-nodev is a pytest plugin that enables *test-driven code search* for Python.

1.3 Test-driven code reuse

Test-driven reuse (TDR) is an extension of the well known *test-driven development* (TDD) development practice.

Developing a new feature in TDR starts with the developer writing the tests that will validate the correct implementation of the desired functionality.

Before writing any functional code the tests are run against all functions and classes of all available projects.

Any code passing the tests is presented to the developer as a candidate implementation for the target feature:

- if nothing passes the tests the developer need to implement the feature and TDR reduces to TDD
- if any code passes the tests the developer can:
  - **import**: accept code as a dependency and use the class / function directly
  - **fork**: copy the code and the related tests into their project
  - **study**: use the code and the related tests as guidelines for their implementation, in particular identifyng corner cases and optimizations

1.4 Unit tests validation

An independent use case for test-driven code search is unit tests validation. If a test passes with an unexpected object there are two possibilities, either the test is not strict enough and allows for false positives and needs to be updated, or the *PASSED* is actually a function you could use instead of your implementation.

1.5 Feature specification tests

Similarly to *keyword-based search* also in *test-driven code search* the quality of the *search results* depends strongly from the ability to build a strong *search query*, in particular from the way our *feature specification tests* are written.

Writing effective *feature specification tests* is an art.

1.6 Bibliography

- "CodeGenie: a tool for test-driven source code search", O.A. Lazzarini Lemos *et al*, Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, 917–918, **2007**, ACM, http://dx.doi.org/10.1145/1297846.1297944
- "Code conjurer: Pulling reusable software out of thin air", O. Hummel *et al*, IEEE Software, (25) 5 45-52, **2008**, IEEE, http://dx.doi.org/10.1109/MS.2008.110 — PDF
- "Finding Source Code on the Web for Remix and Reuse", S.E. Sim *et al*, 251, **2013** — PDF
- "Test-Driven Reuse: Improving the Selection of Semantically Relevant Code", M. Nurolahzade, Ph.D. thesis, **2014**, UNIVERSITY OF CALGARY — PDF

CHAPTER 2

Quickstart

2.1 New user FAQ

nodev-starter-kit lets you perform test-driven code search queries with pytest-nodev safely and efficiently using docker.

Why do I need special care to run pytest-nodev?

Searching code with pytest-nodev looks very much like running arbitrary callables with random arguments. A lot of functions called with the wrong set of arguments may have unexpected consequences ranging from slightly annoying, think `os.mkdir('false')`, to **utterly catastrophic**, think `shutil.rmtree('/', True)`. Serious use of pytest-nodev, in particular using `--candidates-from-all`, require running the tests with operating-system level isolation, e.g. as a dedicated user or even better inside a dedicated container.

But isn't it docker overkill? Can't I just use a dedicated user to run pytest-nodev?

We tried hard to find a simpler setup, but once all the nitty-gritty details are factored in we choose docker as the best trade-off between safety, reproducibility and easiness of use.

2.2 Install nodev-starter-kit

To install *nodev-starter-kit* clone the official repo:

```
$ git clone https://github.com/nodev-io/nodev-starter-kit.git
$ cd nodev-starter-kit
```

Advanced GitHub users are suggested to fork the offical repo and clone their fork.

2.3 Install docker-engine and docker

In order to run pytest-nodev you need to access a docker-engine server via the docker client, if you don't have Docker already setup you need to follow the official installation instructions for your platform:

- Docker for Linux
- Docker for MacOS
- Docker for Windows

Only on Ubuntu 16.04 you can use the script we provide:

```
$ bash ./docker-engine-setup.sh
```

And test your setup with:

```
$ docker info
```

Refer to the official Docker documentation for trouble-shooting and additional configurations.

2.4 Create the nodev image

The *nodev* docker image will be your search engine, it needs to be created once and updated every time you want to change the packages installed in the search engine environment.

With an editor fill the requirements.txt file with the packages to be installed in the search engine.

Build the docker image with:

```
$ docker build -t nodev .
```

2.5 Execute a search

Run the search engine container on a local docker-engine server, e.g. with:

```
$ docker run --rm -it -v `pwd`:/src nodev --candidates-from-stdlib tests/test_parse_bool.py
```

Or alternatively after having set the `DOCKER_HOST` environment variable, e.g. with:

```
$ export DOCKER_HOST='tcp://127.0.0.1:4243'  # change '127.0.0.1:4243' with the IP address and port
                                             # of your docker-engine host
```

you can run the search engine container on a remote docker-engine server, e.g. with:

```
$ python docker-nodev.py --candidates-from-stdlib tests/test_parse_bool.py
======================= test session starts ========================
platform darwin -- Python 3.5.1, pytest-2.9.2, py-1.4.31, pluggy-0.3.1
rootdir: /tmp, inifile: setup.cfg
plugins: nodev-1.0.0, timeout-1.0.0
collected 4000 items

test_parse_bool.py xxxxxxxxxxxx[...]xxxxxxxxXxxxxxxxx[...]xxxxxxxxxxxx

====================== pytest_nodev: 1 passed ======================

test_parse_bool.py::test_parse_bool[distutils.util:strtobool] PASSED

=== 3999 xfailed, 1 xpassed, 260 pytest-warnings in 75.38 seconds ====
```

2.6 Project resources

Documentation	http://nodev-starter-kit.readthedocs.io
Support	https://stackoverflow.com/search?q=pytest-nodev
Development	https://github.com/nodev-io/nodev-starter-kit

2.7 Contributing

Contributions are very welcome. Please see the CONTRIBUTING document for the best way to help. If you encounter any problems, please file an issue along with a detailed description.

Authors:

- Alessandro Amici - @alexamici

Sponsors:

-

2.8 License

nodev-starter-kit is free and open source software distributed under the terms of the MIT license.

CHAPTER 3

Specification Tests

> "If it's not tested, it's broken." - Bruce Eckel.

Testing assure you that the code works... at least in one very specific case.

3.1 Why do we love unit tests?

We love unit tests because they...

- help writing better code in the first place
- make refactoring possible
- keep internal API tidy
- help design and document the intended behaviour of the code

3.2 Why do we hate unit tests?

We hate unit tests because they...

- need as much work as code
- need to be refactored during a refactoring
- break when you change trivial implementation details
- risk keeping the focus on the process, not on the product

3.3 Feature vs. implementation

3.4 How to test for a feature without knowing the implementation?

3.5 Examples

Another example, find a function that decomposes a URL into individual rfc3986 components:

```
$ py.test examples/test_rfc3986_parse.py --candidates-from-modules urllib.parse
[...]
examples/test_rfc3986_parse.py::test_rfc3986_parse_basic[urllib.parse:urlparse] HIT
examples/test_rfc3986_parse.py::test_rfc3986_parse_basic[urllib.parse:urlsplit] HIT
[...]
```

the two functions `urlparse` and `urlsplit` pass the basic rfc3986 parsing test, but do not pass the more complex `test_rfc3986_parse_full` test.

More advanced functions are available on PyPI:

```
$ pip install urllib3
$ py.test examples/test_rfc3986_parse.py --candidates-from-modules urllib3
[...]
examples/test_rfc3986_parse.py::test_rfc3986_parse_basic[urllib3.util.url:parse_url] HIT
examples/test_rfc3986_parse.py::test_rfc3986_parse_full[urllib3.util.url:parse_url] HIT
[...]
```

now the function `parse_url` in the module `urllib3.util.url` passes both tests.