Tools

Notes on tools organized by tool domain.

General

Articles: - Open sourcing a Python project the right way

Version control

Git

The way to use git or the git-flow.

Documentation

Sphinx & Read the Docs

Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, written by Georg Brandl and licensed under the BSD license.

Read the Docs automates the process of building and uploading Sphinx documentation after every commit.

https://readthedocs.org/

Of course, this site is also created from reStructuredText sources using Sphinx! The following features should be highlighted:

Output formats: HTML (including Windows HTML Help), LaTeX (for printable PDF versions), ePub, Texinfo, manual pages, plain text Extensive cross-references: semantic markup and automatic links for functions, classes, citations, glossary terms and similar pieces of information Hierarchical structure: easy definition of a document tree, with automatic links to siblings, parents and children Automatic indices: general index as well as a language-specific module indices Code handling: automatic highlighting using the Pygments highlighter Extensions: automatic testing of code snippets, inclusion of docstrings from Python modules (API docs), and more Contributed extensions: more than 50 extensions contributed by users in a second repository; most of them installable from PyPI

http://sphinx-doc.org/tutorial.html

Software testing

Python unit testing framework

PyUnit or the Python unit testing framework is the de facto standard unit testing framework for Python. It is based on JUnit.

PyUnit forms a part of the Python Standard Library as of Python version 2.1.

Python unit testing in nutshell: - define test in advance beforemaking the main function - define function as normal - make new python script and import unittest - use unit test like this:

```
```
class MyTest(unittest.TestCase):
    def test(self):
        self.assertEqual(my_funcion(my_parameter), exected value)
        self.assertEqual(squareNuber(2),5) #false
        self.assertEqual(squareNuber(2),5) #true
```
```

run script and compare results ### PyTest

pytest is a mature full-featured Python testing tool that helps you write better programs.

Tox

Tox aims to automate and standardize testing in Python. It is part of a larger vision of easing the packaging, testing and release process of Python software.

Tox is a generic virtualenv management and test command line tool you can use for: - checking your package installs correctly with different Python versions and interpreters - running your tests in each of the environments, configuring your test tool of choice - acting as a frontend to Continuous Integration servers, greatly reducing boilerplate and merging CI and shell-based testing.

Robot Framework

Robot Framework is a generic test automation framework for acceptance testing and acceptance test-driven development (ATDD). It has easy-to-use tabular test data syntax and it utilizes the keyword-driven testing approach. Its testing capabilities can be extended by test libraries implemented either with Python or Java, and users can create new higher-level keywords from existing ones using the same syntax that is used for creating test cases.

Continuous integration

Travis

Travis is the home of open source testing. Over 200k open source projects and 126k users are testing on Travis CI. Easily sync your GitHub projects with Travis CI and you'll be testing your code in minutes!

CircleCi

Let CircleCi help your team focus on making a great product. Speed up your testing and development cycle to improve productivity. CircleCI is flexible to run in your environment and scale with your growth. Have the peace of mind by reducing bugs and improving the quality of your application.

- Fast, automatic parallelism and intelligent test splitting
- Quick & Easy Setup
- View build status from GitHub or see PRs and commit messages from CircleCI.
- Beautiful User Experience
- Real-time Build Output and Detailed Test Failure Data
- Smart Notifications
- Inferred Test Commands and Pre-installed Packages and Services
- SSH Access
- Start free and scale without limit
- Free for OSS
- YAML-based Config
- RESTful API and Webhooks
- Build Artifacts
- Sudo Support
- iOS and Android
- Docker
- Deployment Keys and Secrets

Backlog management

General

- Sprint template
- How to maintain a lean product backlog
- Working with storyless tasks

Below some important points related to backlog management. Some content and quotes taken from MountainGoatSoftware.com and ScrumAlliance.org.

Quote:

```
If you don't get the user needs right, it doesn't matter how well you
execute the rest of the project. The Standish Group surveyed IT executive
managers in 1995 for their opinions about why projects succeed.
The three major reasons that a project will succeed are
user involvement (15.9%), executive management support (13.9%), and a clear
statement of requirements (13%). Opinions about why projects are impaired
and ultimately canceled included incomplete requirements (13.1%) and lack
of user involvement (12.4%) at the top of the list. In his now classic ROI
presentation at XP 2002, Jim Johnson shared research regarding the
productivity achieved by those software projects that do deliver
something. Among these, he found that 45% of product features are never
used, 19% are rarely used, and 36% are routinely used. This tells us that
traditional requirement engineering practices are not helping us deliver
value to our customer through our traditional practices.
```

The goals described in the product vision must be broken down into clear sub goals and requirements. Backlogs are used for managing these sub goals and requirements.

User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system:

```
*As a <type of user>, I want <some goal> so that <some reason>.*
```

or

```
*As a <type of user>,
I want <to perform some task>
so that I can <achieve some goal/benefit/value>.*
```

When user stories are too large to be completed in one iteration they are called *epics*. They must be split into multiple smaller user stories before they can be worked on. Example:

```
*As a user, I want to monitor ongoing experiments so that I know what to do
next.*
```

It should be broken down to:

As a user, I want to monitor prespecified variable values of ongoing experiments so that I next.
...

Note:

The risk of writing stories too detailed is that you minimize the involvement of the team, if you write stupid things you get stupid functionality, but no stupid questions. Thus spend less time on defining what stories should look like and spent more time on involving the PO with the team.

Agilefant

Agilefant is an end-to-end project management tool that supports also product planning and portfolio management.

- it is a hierarchical backlog tool: *Product* > *Project* > *Iteration*, and *Story* > *Task*
- *iteration burndown* is a graphical representation of tasks' progress in the iteration as a function of time: It shows a linear reference based on the sum of the tasks' *original estimates* (stories must be split into estimated tasks).
- stories without child stories are called *leaf stories*
- stories can be anything from business goals, customer requests and features to small issue descriptions
- a story consists of zero or more tasks but tasks can also be unlinked to any story
- tasks are used to describe the work needed to get the story done or some small tasks such as meeting or phone call
- a story must always reside in a product, project or an iteration
- stories in the product backlog have no particular *priority*
- a story can then be assigned to a project and/or an iteration where it can be prioritized seperately for both
- story states:
  - *Done* – it reflects the DoD and is thus considered in relevant metrics
  - *Deferred* – the story has been decided to be skipped in this iteration; the *effort left* / *points* are omitted in all metrics; another option is to move the story to another project/iteration
  - *Not Started* – No work has yet been put into realizing this story
  - *In Progress* – ongoing and some work has already been put in
  - *Pending* – waiting for something external that can reasonably be expected to happen without us taking any further action

  - *Blocked* – can't proceed; most likely some action must be taken by 'us' before work can proceed
  - *Ready* – done if only some relatively minor DoD criteria gets met: peer review
- stories may also be assigned a *business value* to facilitate sprint planning
- stories can be *labeled*, f. ex: bugs, usability improvements, strategic functionality, and so on
- stories can be set to depend on other stories
- stories can have attachments (images, files)
- a task is something relatively well-defined and effort-wise small that needs to get done
- tasks are estimated in man-hours; *effort left* refers to the hours left to get the task done. A story's *effort left* is the sum from its tasks. A task's first effort estimate is its *original estimate* which affects the iteration burndown's estimate line.
- tasks also have *spent effort*, the effort already worked on it
- if you add spent effort entries within 8h of each other, the tiem difference is given as default when logging the next entry
- users can be set as responsible for stories and tasks to indicate who is working on what, note that task responsibilities are not propagated upwards to stories

Resources

- FAQ
- User guide
- Introduction and usage example

Trello

- Scrum for Trello
- 10 tips for using Trello for Scrum
- 5 tips for using Trello for Scrum
- Trello-Flowdock integration

Collaborative editing

Floobits & Sublime

- Provides realtime simultaneous file editing collaboration cababilities.
- Can be setup with a repo such that changes are easy to reflect back in GitHub.

See http://awan1.github.io/subl-floo-tutorial.html