Organization of small and medium sized Python projects

Roman Pavelka, 2017-10-26

Motivation for this talk

- Less mistakes
- Less frustration
- Shorter time to deliver
- Higher quality of resulting product

Outline

- General principles
- Source code organization
- Tips and tricks
- Documentation
- Testing
- Further resources

General principles: Primary Technical Imperative

Software's primary technical imperative is managing complexity.

Steve McConnell, *Code Complete*

General principles: KISS

Keep it simple stupid.

Kelly Johnson

It seems that perfection is reached not when there is nothing left to add, but when there is nothing left to take away.

Antoine de Saint-Exupéry

General principles: Process

My approach:

- Define requirements and threats
- Design the system
  - Design should address all requirements and threats
- Construct the system

Doesn’t sound too agile, right? Well...

General principles: Standards

- Agree on standards and rules
  - Time formats and zones, language, style (PEP-8, PEP-257), git
- Follow what has been agreed on
  - Discipline required, avoid exceptions by laziness
- Use automatic checkers and commit hooks
- Code reviews have surprisingly significant value

Source code organization: Top level first

```
import argparse

def main():
    # argparse and config parse, no logic
    well_named_top_level_function(args, config_values)

def well_named_top_level_function(args, config_values):
    component_abc()

def component_abc():
    return 123

If __name__ == main():
    main()
```

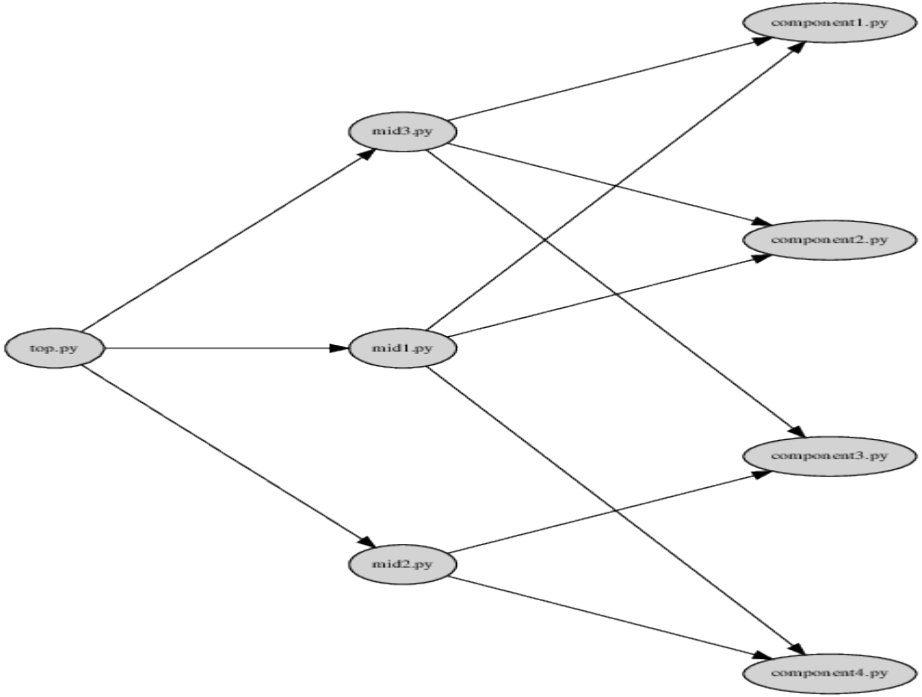
Source code organization: On objects

- **WARNING: controversial topic**
- Objects are overused
- Inheritance is overused
- Recognize when they do bring a value

Source code organization: Three layers file hierarchy

- Project specific components
- Domain / problem specific components
- Building blocks, general-purpose reusable components

Source code organization: Three layers file hierarchy



Tips and tricks

- snakefood
- flake8 --ignore=W503
- Avoid related stuff in different places of the code
  - document if unavoidable
- Avoid side-effects
  - e.g. beware too clever setters and getters
- A pipeline is the best topology
- Name things well, explain your intent in code

Documentation: General remarks

- Documentation requires discipline
- Keep it as close to code as possible
- Make change -> Reflect it in documentation
- Start with answering “Why?”
  - A great talk by Simon Sinek “Start With Why”
- Code is documented best by docstrings and comments
  - Sphinx then generates nice documentation

Documentation: Very simple programs

- README.md often suffice for user, bare minimum to provide:
  - Purpose
  - Dependencies and installation
  - Usage
  - How to contribute (if applicable)
  - Contact to maintainer
- In-code comments and docstrings for developers
  - describe what is not obvious

Documentation: more complex programs

- Provide README.md or similar with dependencies, installation and usage as well
- Provide section with requirements and threats
- Provide design document
    - Describes environment, components and their interfacing
- Auto-generated developer’s guide is nice to have addition

Testing: Simple Testing Can Prevent Most Critical Failures

- A majority (77%) of the failures require more than one input event to manifest, but most of the failures (90%) require no more than 3.
- Almost all (98%) of the failures are guaranteed to manifest on no more than 3 nodes. 84% will manifest on no more than 2 nodes.
- 74% of the failures are deterministic — they are guaranteed to manifest given the right input event sequences.
- A majority of the production failures (77%) can be reproduced by a unit test.
- Almost all catastrophic failures (92%) are the result of incorrect handling of non-fatal errors explicitly signaled in software.
- 35% of the catastrophic failures are caused by trivial mistakes in error handling logic — ones that simply violate best programming practices and that can be detected without system specific knowledge.

From: Yuan et al. (2014), Simple Testing Can Prevent Most Critical Failures

Testing: Test-driven development

- Write a test first, then code under the test itself
- Hard to test = Maybe badly designed
- Extensive testing leads to surprising architectural improvement
- Lot of unittest, much less integration and system tests
- Google Testing Blog: Just Say No to More End-to-End Tests
  - Value of test, flaky tests, testing pyramide
- Test a lot, test often, test both happy and sad paths

Further resources

- Fred Brooks, 1995: Mythical Man-Month
- McConnell, 2004: Code Complete
- Uncle Bob Expecting Professionalism
  - Robert “Uncle Bob” Martin

Outline

- General principles
- Source code organization
- Tips and tricks
- Documentation
- Testing
- Further resources

Available at: romanpavelka.cz/pyvo.pdf

Thank you for attention!

Any questions?