Software Engineering

Slides by Carl Kingsford

Apr. 2, 2013

The Most Important Rules

- **Use** `asserts`:

```
def parent(self, pos):
    """Return the position of the parent of pos"""
    assert pos >= 0 and pos < len(self.heap)
    if pos == 0: return None
    return int(math.ceil(pos / float(self.ARITY)) - 1)
```

- **Comment code**, but avoid pointless comments:

```
# get the current estimate for g(v)    [good]
vi, gv = Visited[v] if v in Visited else (None, Infinity)

# return gv        [bad]
return gv
```

- **Function follows form**: make your code readable from the start. "I'll clean it up when it works" is a poor practice.

The Most Important Rules, II

- **Avoid code duplication**: do not copy and paste code (not even if you are in a hurry). Make a function that encapsulates what you are trying to do.
- **Don't use global variables**. Some exceptions:
  - options for your "main routine"
  - the "main" data structure you are working on (avoids passing it to every function)
- **The maximum line length is 80 characters** (even in Python).

  80 characters is about the reasonable amount that you can understand at a time. Python automatically merges lines that span () or {}, etc.
- **The TAB character is an evil abomination** with no point and no place in any file you ever create (except Makefiles).

The Most Important Rules, III

- **Write small functions** ($<$ 25 lines)

Some Particulars for Scientific Programming

- **Fail gracefully but immediately** (write fragile code).

  Do not try to "fix" up the input or guess the user's intent. If it looks wrong, assume it's wrong and just `exit(3)`.

  For Microsoft Word, there's some utility it never crashing. For scientists, a crash is better than a wrong answer.

- Sometimes the right answer isn't known, making debugging hard. So: **implement two versions of important and tricky routines**.

- **Don't do black box testing**: don't just compare to expected outputs, check after each major operation that the data makes sense.

Scientific Programming, II

- Don't use or require many command line options.
  Real email I got in Feb:

  ```
  Run as following:
  python cefercplex.py -t 0 -m 0 -d 0 -f mytraces
  -g contact.gml -s 1 -sparam 1.0 -samp 0
  -n 0.0 -e 0 -a 1 -p 0 -sprob 0.15
  ```

  This hurts reproducibility. Instead (1) eliminate options that are not useful (don't over generalize!) and (2) use configuration files.

- Don't hard code any paths or names of data files.

- Don't store results in thousands of intermediate files. (Use Pickle or a simple database).

Design

- **Use object oriented design but use it sparingly**: objects should represent the "things" you are manipulating in your program.
  Don't be overly general.

- **Don't use bad languages**: avoid Awk, Perl, Make, Shell for anything other than 1-liners or tiny scripts.

- **Use existing libraries** (STL, boost, python standard library, libraries in the Python cheese shop, etc.) before writing your own. (But only use well-tested libraries)

Style

- decide on a consistent naming scheme and style (e.g. Google Style Guide)
- don't use magic numbers, use constants
- don't include useless comments
- clarity often comes from brevity
- don't be needlessly clever
- don't use edge cases of the language: `array[i++] = i;` is in fact undefined in C.

Build up a program a bit at a time

- write and test code in small chunks
- use unit tests
- fix bugs as you go, not at the end
- use namespaces, modules, etc. to isolate large chunks of code

Unit tests in Python

See `http://docs.python.org/2/library/unittest.html`.

Optimization

- Write first for correctness, then for clarity, FINALLY for efficiency
- Only optimize at the end when you find what is really slow
- Use the Python profiler (or gprof for C/C++)

Python Profiler

Very easy to use.

Usually just need to add the blue part in front of your program:

```
python -m cProfile -s cumulative astar.py tsp-input/r30em1.gexf
```

Can add "`-o filename`" to save the profile to a file and then use:

```
import pstats
p = pstats.Stats('filename')
p.sort_stats('cumulative').print_stats(10)
```

to manipulate all the saved data about your program's run.

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
2	0.002	0.001	1.446	0.723	astar.py:3(<module>)
1	0.000	0.000	1.370	1.370	astar.py:210(main)
1	0.003	0.003	1.357	1.357	astar.py:124(astar)
842	0.004	0.000	1.341	0.002	astar.py:226(<lambda>)
842	0.011	0.000	1.336	0.002	astar.py:179(msth)
842	0.009	0.000	1.169	0.001	mst.py:87(minimum_spannin
1686	0.003	0.000	1.160	0.001	graph.py:178(__init__)
843	0.002	0.000	1.157	0.001	convert.py:73(to_networkx
842	0.001	0.000	1.150	0.001	convert.py:391(from_edgel
1277	0.044	0.000	1.148	0.001	graph.py:722(add_edges_fr
17140	0.224	0.000	1.089	0.000	mst.py:22(minimum_spannin
403796	0.495	0.000	0.567	0.000	union_find.py:40(__getite
842	0.072	0.000	0.137	0.000	{sorted}
1684	0.025	0.000	0.127	0.000	graph.py:1007(edges)
842	0.104	0.000	0.125	0.000	graph.py:1445(subgraph)
203582	0.088	0.000	0.102	0.000	graph.py:1052(edges_iter)
16298	0.051	0.000	0.094	0.000	union_find.py:65(union)

Social & Group Management

- Use version control (git, subversion, sourceforge, github).
  Do not check in automatically created files!

- Use code reviews: force one group member to explain their code, at both a high level and line-by-line to another group member.

- Don't change code that isn't broken (to fix style, etc.)

- Don't "flip the bozo bit" (McCarthy).

Agile Software Development

Beck et al. (`http://agilemanifesto.org`)

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - *Individuals and interactions over processes and tools*
> - *Working software over comprehensive documentation*
> - *Customer collaboration over contract negotiation*
> - *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

Principles behind the Agile Manifesto (Beck et al).

URL: `http://www.agilemanifesto.org/principles.html`

"We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project."

"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."