Ashish Mahabal
California Institute of Technology

Best Programming Practices - II

Sponsored by the Jet Propulsion Laboratory and by the Keck Institute for Space Studies, California Institute of Technology

Before the project

Dig for requirements

Document requirements

Make use case diagrams

Maintain a glossary

Document, Document, …

Easy development versus easy maintenance

- projects live much longer than intended
- adopt more complex and readable language

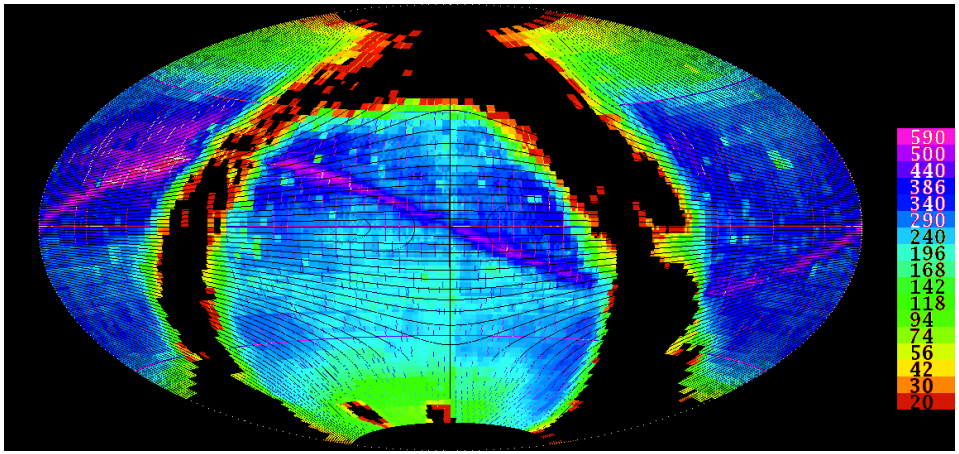
Check requirements

Design, implement, integrate

Validate

Validation

- Don’t trust the work of others
  - Validate data (numbers, chars etc.)
  - Put constraints (-90 <= dec <= 90)
  - Check consistency



Validation

- Don’t trust the work of others
  - Validate data
  - Put constraints
  - Check consistency
- Don’t trust yourself
  - Do all the above to your code too

When something goes wrong

- Crash early
  - Sqrt of negative numbers (require, ensure, NaN)
- Crash, don't trash
  - Die
  - Croak (blaming the caller)
  - Confess (more details)
  - Try/catch (own error handlers e.g. HTML 404)
- Exceptions – when to raise them
  - should it have existed?
  - Don't know?

try/except

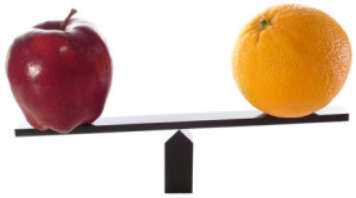
Yes:

```
try:
    value = collection[key]
except KeyError:
    return key_not_found(key)
else:
    return handle_value(value)
```

No:

```
try:
    # Too broad!
    return handle_value(collection[key])
except KeyError:
    # Will also catch KeyError raised by handle_value()
    return key_not_found(key)
```

- Don’t optimize code – benchmark it
- Don’t optimize data structures – measure them
- Cache data when you can – use Memoize
- Benchmark caching strategies
- Don’t optimize applications – profile them (find where they spend most time)



gridgain.blogspot.com

Momoization

```
factorial_memo = {}
def factorial(k):
    if k < 2: return 1
    if not k in factorial_memo:
        factorial_memo[k] = k * factorial(k-1)
    return factorial_memo[k]

factorial(10)
```

Profiling

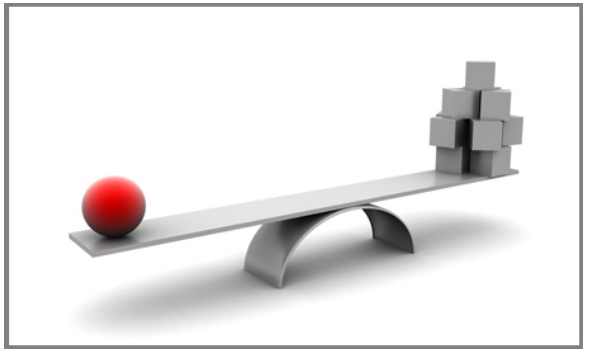
```
import cProfile
import re
cProfile.run('re.compile("Hello|World")')
```

```
         238 function calls (233 primitive calls) in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 re.py:188(compile)
        1    0.000    0.000    0.000    0.000 re.py:226(_compile)
        1    0.000    0.000    0.000    0.000 sre_compile.py:178(_compile_charset)
        1    0.000    0.000    0.000    0.000 sre_compile.py:207(_optimize_charset)
        4    0.000    0.000    0.000    0.000 sre_compile.py:24(_identityfunction)
      3/1    0.000    0.000    0.000    0.000 sre_compile.py:32(_compile)
        1    0.000    0.000    0.000    0.000 sre_compile.py:361(_compile_info)
        2    0.000    0.000    0.000    0.000 sre_compile.py:474(isstring)
```

Benchmarking

Benchmarking game:

http://shootout.alioth.debian.org/



blog.insresearch.com

Benchmarking python:

http://ziade.org/2007/10/18/unobtrusive-benchmark-and-debug-of-python-applications/

Necessary ingredients

- Robustness
- Efficiency
- Maintainability

Robustness

- Introducing (tests for) errors
  - checking for existence (uniform style)
- Edge cases
  - 0? 1? last?
- Error handling
  - exceptions? Verifying terminal input
- Reporting failure
  - Traces? Errors don't get quietly ignored

Checking for overloaded cases

```
def square(x):
    """Squares x.

    >>> square(2)
    4
    >>> square(-2)
    4
    >>> square(complex(0,1))
    (-1+0j)
    """

    return x * x

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Efficiency

- Working with strength
- Proper data structures
- Avoiding weaknesses
- Dealing with version changes (backward compatibility) [python 2.X and 3.0!]

Maintainability

- More time than writing
- You don’t understand your own code
  - Comment amply
- You yourself will maintain it
- Consistent practices
  - Braces, brackets, spaces
  - Line lengths, tabs, blank lines

Next time …

- Design by contract
- Comments, Arguments and all that

http://ib.ptb.de/8/85/851/sps/swq/graphix