Language alone won’t pay your bills

Alan Franzoni - EP 2012
twitter: franzeur
website: www.franzoni.eu
What’s this about?
What’s this about?

• Original idea: “Why Python sucks”
What’s this about?

• Original idea: “Why Python sucks”

• What I really wanted to say: are you aware of your tradeoffs?
What’s this about?

• Original idea: “Why Python sucks”

• What I really wanted to say: are you aware of your tradeoffs?

• Do you know why you should not use Python in a certain context?
What’s your aim?
What’s your aim?

• Should you enjoy? (maybe)
What’s your aim?

- Should you enjoy? (maybe)
- Should your software be fast? (define fast) (maybe)
What’s your aim?

- Should you enjoy? (maybe)
- Should your software be fast? (define fast) (maybe)
- Development productivity
Software total ownership costs
Software total ownership costs

You should consider the total cost in order to deliver your software to your customer(s) and to maintain it working.
Main topics
Main topics

• Language
Main topics

- Language
- Code reuse
Main topics

• Language
• Code reuse
• Tools
Main topics

• Language
• Code reuse
• Tools
• Deployment
Language
Language

• Python is pretty good
Language

• Python is pretty good
• Dynamically typed
package eu.franzoni.ep2012;

import eu.franzoni.ep2012.impl.FunnyDuck;

import java.util.Collection;

public class MyDuckGenerator {
    public Duck createDuck(Flesh flesh, Collection<Wing> wings) {
        /* ... */
        return new FunnyDuck();
    }
}
class MyDuckGenerator(object):
    def createDuck(self, flesh, wings):
        """
        :type flesh: :class: python_things.flesh.Flesh
        :type wings: :class: collections.Iterable
        :rtype: :class:python_things.ducks.FunnyDuck
        """
        # some code here
        return FunnyDuck()
Typing is not the bottleneck
Typing is not the bottleneck

Typing IS NOT THE BOTTLENECK
Typing is not the bottleneck

(C) Sebastian Hermida    www.sbastn.com
It's not static typing that gives Java its slow-and-tedious feel.
sys.path
sys.path

vs
sys.path

vs

classpath
TRADEOFF: you’re trading power and freedom for a recognized way of doing something
Code reuse 1: libraries

- Standard lib gives you **quick and full access** to the underlying OS api, but may limit portability.

- Libraries may be linked to C libraries -> you can reuse existing C code, but **deployment behaviour** may vary.
Code reuse 2 - packaging
Code reuse 2 - packaging

• Distutils
Code reuse 2 - packaging

- Distutils
- Setupptools
Code reuse 2 - packaging

- Distutils
- Setuptools
- Distribute
Code reuse 2 - packaging

- Distutils
- Setuptools
- Distribute
- Distutils2
Code reuse 2 - packaging

- Distutils
- Setuptools
- Distribute
- Distutils2
- Pip
Code reuse 3 - isolation
Code reuse 3 - isolation

- zc.buildout
- virtualenv
Diamond dependency
Play together?
Python Packaging

• Strange things done in setup.py, even importing a module before installing it
• Some package working with distribute, others with setuptools
• Missing dependencies or version conflicts
• Mutable PyPI -> needs mirroring
FACTOID #1

Not working packages from pypi
FACTOID #1

Not working packages from pypi

33%
FACTOID #2
FACTOID #2

• Java project with 50-60 deps
FACTOID #2

• Java project with 50-60 deps
• We had to fork one project because of a subtle bug
FACTOID #2

• Java project with 50-60 deps
• We had to fork one project because of a subtle bug
• Python project with about 10 deps
FACTOID #2

• Java project with 50-60 deps
• We had to fork one project because of a subtle bug
• Python project with about 10 deps
• We had to fork five libraries because of packaging issues or version conflicts
Maven
Maven

- Its XML is a nightmare to newcomers
Maven

- Its XML is a nightmare to newcomers
- It’s declarative
Maven

• Its XML is a nightmare to newcomers
• It’s declarative
• You basically get the very same build on all machines
Maven

- Its XML is a nightmare to newcomers
- It’s declarative
- You basically get the very same build on all machines
- Proxy/caching repositories are available
A matter of authority?
A matter of authority?

- Python core is not concerned with too many tools, as they aren’t directly connected to the language
A matter of authority?

- Python core is not concerned with too many tools, as they aren’t directly connected to the language
- But they’re needed whatsoever
A matter of authority?

• Python core is not concerned with too many tools, as they aren’t directly connected to the language

• But they’re needed whatsoever

• There’s nothing like Apache or Eclipse for Python. Individual developers write and rewrite solutions.
Why code reuse matters
Why code reuse matters

- If somebody else has spent years and years in development, should you care and rebake your own solution?
Why code reuse matters

- If somebody else has spent years and years in development, should you care and rebake your own solution?
- Don’t fall for the NIY syndrome!
Why code reuse matters

• If somebody else has spent years and years in development, should you care and rebake your own solution?

• Don’t fall for the NIY syndrome!

• Code reuse in Java is much easier
Why code reuse matters

• If somebody else has spent years and years in development, should you care and rebake your own solution?

• Don’t fall for the NIY syndrome!

• Code reuse in Java is much easier

• **Tradeoff:** it may be quicker and more fun to write in Python, but reusing other’s and your own code may be harder!
TOOLS
Tools matter
Tools matter

• A good IDE just breaks any vanilla Emacs or Vim on functionality
Tools matter

- A good IDE just breaks any vanilla Emacs or Vim on functionality
- Learning curve is usually better for newcomers
Tools matter

- A good IDE just breaks any vanilla Emacs or Vim on functionality
- Learning curve is usually better for newcomers
- Code-completion can help you a lot
Tools matter

• A good IDE just breaks any vanilla Emacs or Vim on functionality

• Learning curve is usually better for newcomers

• Code-completion can help you a lot

• A good debugger can help you dig into complex and tricky situations.
Tools matter

• A good IDE just breaks any vanilla Emacs or Vim on functionality

• Learning curve is usually better for newcomers

• Code-completion can help you a lot

• A good debugger can help you dig into complex and tricky situations.

• Refactoring matters a lot.
Python tools miss integration
Python tools miss integration

- Good IDE (IMHO) Pycharm tries to use ‘default’ tools
- Other IDEs setup their own build files which are hard to use without the IDE (e.g. CI)
- Debugger integration is really tricky
Java tools work together
Java tools work together

• Highly integrated
Java tools work together

- Highly integrated
- Just pull in a pom.xml and your project is setup, including paths, dependencies, code completion, deployment to an application server
Java tools work together

- Highly integrated
- Just pull in a pom.xml and your project is setup, including paths, dependencies, code completion, deployment to an application server
- It’s very fast to pickup on an existing project and start hacking
Java tools work together

- Highly integrated
- Just pull in a pom.xml and your project is setup, including paths, dependencies, code completion, deployment to an application server
- It’s very fast to pickup on an existing project and start hacking
- This is quite true to Ruby as well. Gems work fine.
CONTINUOUS INTEGRATION AND DELIVERY
DELIVERY
DEVELOPMENT

- Deployments are an highly manual task in Python, both for webapps and standalone apps
- There’s no recommended way to do it.
- Packaging into wholly contained directories for deb/rpms is all but obvious and requires handcrafing.
Continuus Integration
Continuos Integration

- Need for reproducible builds
- Hard to make an environment stable and deliver the very same artifact that was built to production

• *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, by J. Humble, D. Farley, Addison-Wesley 2010

• *Code Complete: A Practical Handbook of Software Construction*, by S. McConnell, Microsoft Press 2004