From flop to success in academic software development

Gaël Varoquaux
“Most lines of code written by programmers in academia never reach an audience”

G. Varoquaux, March 19th 2015
“Most lines of code written by programmers in academia never reach an audience”

G. Varoquaux, March 19\textsuperscript{th} 2015

- Technical problems: making software
- Marketing problems: unknown users
Choose your battles

Win them

projects that solve a problem

software production

* Too Long, Didn’t Read
Please allow me to introduce myself
I’m a man of wealth and taste
I’ve been around for a long, long year

Physicist gone bad
Neuroscience, Machine learning

Worked in a software startup
Enthought: scientific computing consulting in Python

Coder (done my share of mistake)
Mayavi, scikit-learn, joblib...

Scipy community
Chair of scipy and EuroScipy conferences

Researcher (PI) at INRIA

G Varoquaux
Please allow me to introduce myself

I’m a man of wealth and taste
I’ve been around for a long, long year

**Scikit-learn**
Reference machine-learning package
Installed on 1% of the computers running Debian
16 books on Amazon

**Mayavi**
Reference 3D plotting in Python
Installed on .5% of the computers running Debian

**joblib**
Backend library
Installed on 1.5% of the computers running Debian
Software for scientific research
Reproducible science: enabling falsification

**Replicating**
A 3\textsuperscript{rd} party redoing the work
Code and data made available

**Reproducing**
New analysis on different data / code coming to the same conclusion

**Reusing**
Applying the approach to a new problem

Let us enable reusable research
Reproducible science: enabling falsification

**Replicating**
A 3rd party redoing the work
Code and data made available

**Reproducing**
New analysis on different data / code coming to the same conclusion

**Reusing**
Applying the.

Let us enable reusable research

**Arguments for BSD license**
No strings attached
Can tinker with it
The advancement of knowledge

Imagine a circle that contains human knowledge

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

By the time you finish elementary school, you know a little

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

High school takes you a little bit further

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

With a bachelors degree, you gain a speciality

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

A master’s degree deepens this speciality

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge
Research papers take you to the edge of human knowledge

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

Once you are at the boundary, you focus

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

You push at the boundary for a few years

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

And one day it yields

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

That dent you’ve made, is called a PhD

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

Of course, the world looks different to you now

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

But don’t forget the big picture

Courtesy of Matt Might, via Stefan van der Waalt
The advancement of knowledge

This is an optimistic view

- Physics
- Computer science
- Maths
- Biology
- Economy
- History
- Literature
The advancement of knowledge

This is an optimistic view

I want to be there

Physics
Computer science
Maths
Biology
Economy
History
Literature

G Varoquaux
Translationnal computational science

**Computational science**

The use of computers and mathematical models to address scientific research
Translationnal computational science

**Computational science**
The use of computers and mathematical models to address scientific research

**Translationnal science**
In medicine: bring bench science to medical practice

G Varoquaux
Translationnal computational science

Computational science
The use of computers and mathematical models to address scientific research

Translationnal science
In medicine: bring bench science to medical practice

Translational computational science?
Pick a problem to work on

Take the “easy” route
There needs to be a market screaming for the software (in academia and in industry)

Refine your vision

Pull, not push
Design driven be need
Having an impact
Having an impact
Project idea

A software implementing:

1) machine learning

and 2) neuroimaging

and 3) a graphical user interface

and 4) 3D plotting
Pick the right battles: viable projects

**Project idea**

A software implementing:

1. machine learning
2. neuroimaging
3. a graphical user interface
4. 3D plotting

Define project scope and vision

Break down projects by expertise

Don't solve hard problems

Know the software landscape

Don't target markets that will not yield contributors

Need a vision = elevator pitch

Your research (PhD) probably does not qualify

ñ need to cherry-pick contributions
Pick the right battles: viable projects

Define project scope and vision

- Break down projects by expertise
- Don’t solve hard problems
- Know the software landscape
- Don’t target markets that will not yield contributors

Need a vision = elevator pitch
Pick the right battles: viable projects

Define project scope and vision

- Break down projects by expertise
- Don’t solve hard problems
- Know the software landscape
- Don’t target markets that will not yield contributors

Need a vision = elevator pitch

Your research (PhD) probably does not qualify
⇒ need to cherry-pick contributions
Open source and community development

Code maintenance too expensive to be alone

scikit-learn ~ 300 email/month
nipy ~ 45 email/month
joblib ~ 45 email/month
mayavi ~ 30 email/month

“Hey Gael, I take it you’re too busy. That’s okay, I spent a day trying to install XXX and I think I’ll succeed myself. Next time though please don’t ignore my emails, I really don’t like it. You can say, ‘sorry, I have no time to help you.’ Just don’t ignore.”
Open source and community development

**Code maintenance too expensive to be alone**

- scikit-learn ~ 300 email/month
- nipy ~ 45 email/month
- joblib ~ 45 email/month
- mayavi ~ 30 email/month

**Your “benefits” come from a fraction of the code**

- Data loading? Maybe?
- Standard algorithms? Nah

**Share the common code**...

...to avoid dying under code

Code becomes less precious with time

And somebody might contribute features
Community development in scikit-learn

Huge feature set: benefits of a large team

Project growth:

- More than 200 contributors
- ~ 12 core contributors
- 1 full-time INRIA programmer from the start

Estimated cost of development: $6 millions

COCOMO model,
http://www.ohloh.net/p/scikit-learn
Communities: many eyes makes code fast

Gilles Louppe @glouppe · Feb 18

Speed improvement from 0.13 to 0.15-git of Random Forests in Scikit-Learn:


G Varoquaux
You need a community
6 steps to a community-driven project

1. Focus on **quality**
2. Build great **docs and examples**
3. Use **github**
4. Limit the technicality of your codebase
5. Releasing and packaging matter
6. Focus on your contributors, give them credit, decision power

http://www.slideshare.net/GaelVaroquaux/scikit-learn-dveloppement-communautaire

G Varoquaux
What’s in a scientific-computing environment
Interaction…
  → script…
  → module…
  ⇝ interaction again…

Consolidation,
progressively

Low tech and short
turn-around times
Choose your weapons

Python, what else?

- Interactive language
- Easy to read / write
- General purpose
Choose your weapons

*Python, what else?*

- Interactive language
- Easy to read / write
- General purpose

*Old virtual machine / compiler*

Younger languages promising (Julia)

but will they get adoption beyond science?

G Varoquaux
Choose your weapons

*Python, what else?*

Use numpy arrays

- scikit-learn
- scikit-image

... 

It's about plugin things together
Software architecture for science

- “Scriptability” is paramount
- In an application: MVC (model, view, controller)

Model
Numerical or data-processing core

View
Output: graphs, or files
Must enable headless use

Controller
Input: dialogs, or an API
Avoid input as files: not expressive

Dialogs should never be far from the code
Dialog generation: traits, IPython widgets

Reactive programming:
Dialogs modify object, and the model updates

Don’t own the main

G Varoquaux
“Scriptability” is paramount

In an application: MVC (model, view, controller)

Model
Numerical or data-processing core

View
Output: graphs, or files
Must enable headless use

Controller
Input: dialogs, or an API
Avoid input as files: not expressive

Dialogs should never be far from the code

In Mayavi: script generation for free

Reactive programming:
dialogs modify object, and the model updates

Don’t own the main

G Varoquaux
You need quality

- Quality will give you users
  Bugs give you bad rap

- Quality will give you developers
  Contribute to learn and improve

- Quality will make your developers happy
  People need to be proud of their work

Do less, do better

Goes against the grant-system incentive
Quality: what & how

**Great documentation**
- Simplify, but don’t dumb down
- Focus on what the user is trying to solve

**Great APIs**
- Example-based development
- If something is hard to explain, rethink the concepts
- Limit the number of different concepts and objects
- Consistency, consistency, consistency

**Good numerics**
- Write tests based on mathematical properties
- When a user finds an instability, write a new test
Quality: what & how

**Great documentation**
- Simplify, but don’t dumb down
- Focus on what the user is trying to solve

**Great APIs**
- Example-based development
- If something is hard to explain, rethink the concepts
- Limit the number of different concepts and objects
- Consistency, consistency, consistency

**Quality enables reuse**
- Beyond mere reproducibility

**Good numerics**
- Write tests based on mathematical properties
- When a user finds an instability, write a new test
Be productive
Be productive

“If you spend too much time thinking about a thing, you’ll never get it done.” — Bruce Lee
Limited resources

**Limited resources are good**

- Need success in the short term, not the long term
- The startup culture: fail fast
  - Quickly identify non-viable projects
- The simplest solution that works is the best
Short cycles, limited ambitions

- Keep coming back to your users
- Release early, release often
Simplicity

Complexity increase superlinearly

[An Experiment on Unit Increase in Problem Complexity, Woodfield 1979]

25% increase in problem complexity

⇒ 100% increase in code complexity
Simplicity

**Complexity increase superlinearly**

[An Experiment on Unit Increase in Problem Complexity, Woodfield 1979]

25% increase in problem complexity

$\Rightarrow$ 100% increase in code complexity

**The 80/20 rule**

80% of the usecases can be solved with 20% of the lines of code

Avoid feature creep
Simplicity

**Complexity increase superlinearly**

[An Experiment on Unit Increase in Problem Complexity, Woodfield 1979]

25% increase in problem complexity

\[ \Rightarrow 100\% \text{ increase in code complexity} \]

**The 80/20 rule**

80% of the usecases can be solved with 20% of the lines of code

Avoid feature creep

**Use objects sparingly**

Don’t use classes for the sake of it
Software engineering
Software development is an industrial process

It’s time to adopt engineering practices

Amateur practices that work for small projects do not scale
Software engineering good practices

- Coding convention, good naming
- Version control

Use git + github

- Unit testing
  If it’s not tested, it’s broken or soon will be.

- Make a package,
  with controlled dependencies and compilation
Things we did right (maybe)
Mayavi: 3D visualization in Python

Success factors
- Building upon VTK  
  Great power
- Component model (UI)
- Internals open to the world
  $\Rightarrow$ from interaction to scripting

Limiting factors
- Building upon VTK  
  A lot of complexity
- Codebase too complex and object-oriented  
  (bound to VTK)
- Users of GUIs do not turn into developers
- Composition is an API killer
Mayavi: 3D visualization in Python

Success factors
- Building upon VTK
- Component model (UI)
- Internals open to the world

⇒ from interaction to scripting

Limiting factors
- Building upon VTK
- Codebase too complex and object-oriented
  (bound to VTK)
- Users of GUIs do not turn into developers
- Composition is an API killer

Great power

A lot of complexity
Parallel for loop

```python
>>> from joblib import Parallel, delayed
>>> Parallel(n_jobs=2)(delayed(sqrt)(i**2) for i in range(8))
... [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
```

- On-demand dispatch to ease memory consumption
- Threading and processes backends
Parallel for loop

```python
>>> from joblib import Parallel, delayed
>>> Parallel(n_jobs=2)(delayed(sqrt)(i**2) for i in range(8))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
```

Memoize pattern

```python
mem = joblib.Memory(cachedir=’.’)  

g = mem.cache(f)  

b = g(a)  # computes a using f  

c = g(a)  # retrieves results from store
```
Success factors

- Simplicity of use
- Patterns we really, really need (pull not push)
joblib: computational workflow patterns

Success factors

- Simplicity of use
- Patterns we really, really need (pull not push)

Limiting factor

- Vision of the project unclear
- Positioning with regards to landscape unclear (parallel computing world fuzzy)
- Tricky code inside
Success factors

- Right project vision
  Machine learning without learning the machinery
  Black box that can be opened
  Right trade-off between "just works" and versatility
  (think Apple vs Linux)

We’re not going to solve all the problems for you
  I don’t solve hard problems

Feature-engineering, domain-specific cases...
  Python is a programming language. Use it.

Cover all the 80% usecases in one package
Success factors

- Right project vision

- High-level programming
  - Optimize algorithms, not for loops
  - Know perfectly Numpy and scipy
  - Use Cython, quad not C/C++
Success factors

- Right project vision
- High-level programming
- Good API design
  - separate data from operations
Success factors

- Right project vision
- High-level programming
- Good API design
  - separate data from operations
  - Object API exposes a data-processing language
  - fit, predict, transform, score, partial_fit
  - Instantiated without data but with all parameters
Success factors

- Right project vision
- High-level programming
- Good API design
- Great community
  - Github + code review
Success factors

- Right project vision
- High-level programming
- Good API design
- Great community
- Great documentation
Success factors

- Right project vision
- High-level programming
- Good API design
- Great community
- Great documentation

Limiting factors

- Tricky numerical code
- Our own success → huge volume
From flop to success in academic software

1. Choose the project well
   - Not all battles can be fought
   - Make sure that there is a market
   - Don’t solve (too many) hard problems

@GaelVaroquaux
From flop to success in academic software

1. Choose the project well

2. Reach a community

Users: market your project
Developers: community-driven development
From flop to success in academic software

1. Choose the project well
2. Reach a community
3. Make good software
   With quality, software engineering
   Usability matters

@GaelVaroquaux
From flop to success in academic software

1. Choose the project well
2. Reach a community
3. Make good software