

# Software Sustainability

The why and the how

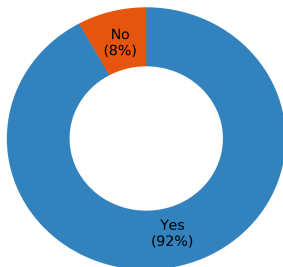
Phil Tooley  
RSE@Sheffield

Cardiovascular Fluids Modelling SIG  
20 September 2018

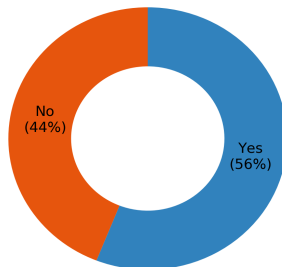
# Science relies on research software

## 2014 UK Research Software Survey<sup>1</sup>

Do you use research software?



Do you develop your own research software?



# It must be *good* research software



## **The war over supercooled water**

How a hidden coding error fueled a seven-year dispute between two of condensed matter's top theorists.

[physicstoday.scitation.org](http://physicstoday.scitation.org)

## Software Sustainability Institute<sup>2</sup>

- ▶ Funded in 2010 by EPSRC to promote sustainable development of research software and now a collaborative effort by multiple funders
- ▶ Focus on reproducible research and recognising software as a research output
- ▶ Encourage widespread recognition of research software engineer (RSE) role as experts in development of high quality software
- ▶ Provide training and support for developers of scientific software through RSEs (software/data carpentry)



<sup>2</sup>[www.software.ac.uk](http://www.software.ac.uk)

# My first research code experience



# My first research code experience

 `README.txt`

```
1 Dear Phil,  
2  
3 This is a copy of the code WAKE, which is written by Mora and Antonsen.  
4 There is also a photon-kinetic version of it, QWAKE. If anyone can read  
5 the comments (largely in French) they are welcome to use it, provided  
6 they properly acknowledge that the code has been handed to them by Luis  
7 Silva (who gave it to me)
```

# My first research code experience

 `README.txt`

```
1 Dear Phil,  
2  
3 This is a copy of the code WAKE, which is written by Mora and Antonsen.  
4 There is also a photon-kinetic version of it, QWAKE. If anyone can read  
5 the comments (largely in French) they are welcome to use it, provided  
6 they properly acknowledge that the code has been handed to them by Luis  
7 Silva (who gave it to me)
```

How do I run this?

What will it do?

How do I interpret the output?

# The code I actually used

README.md

## Fourier-Bessel Particle-In-Cell code (FBPIC)

master passing dev passing pypi v0.9.4 license BSD-3-Clause-LBNL DOI 10.5281/zenodo.1285993

Online documentation: <http://fbpic.github.io>

Support: [Join slack](#)

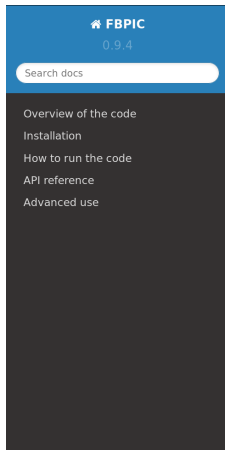
### Overview

FBPIC is a [Particle-In-Cell \(PIC\) code](#) for relativistic plasma physics.

It is especially well-suited for physical simulations of **laser-wakefield acceleration** and **plasma-wakefield acceleration**, with close-to-cylindrical symmetry.



# The code I actually used



## FBPIC documentation

FBPIC (Fourier-Bessel Particle-In-Cell) is a [Particle-In-Cell \(PIC\) code](#) for relativistic plasma physics. It is especially well-suited for physical simulations of **laser-wakefield acceleration** and **plasma-wakefield acceleration**.

The distinctive feature of FBPIC, compared to *most* other PIC codes, is to use a **spectral cylindrical representation**. This makes the code both **fast** and **accurate**, for situations with **close-to-cylindrical symmetry**. For a brief overview of the algorithm, its advantages and limitations, see the section [Overview of the code](#).

In addition, FBPIC implements several **useful features for laser-plasma acceleration**, including:

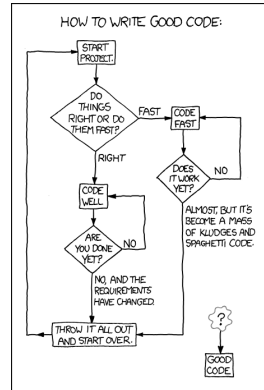
- Moving window
- Calculation of **space-charge fields** at the beginning of the simulation
- Intrinsic **mitigation of Numerical Cherenkov Radiation (NCR)** from relativistic bunches
- **Field ionization** module (ADK model)
- Support for **boosted-frame simulations** (see [Running boosted-frame simulations](#))

FBPIC can run on **multi-core CPU** (with multi-threading) or **GPU**. For large simulations, running the code on GPU can be much faster than on CPU.

# Unsustainable Software

Symptoms of unsustainable software or development practices include:

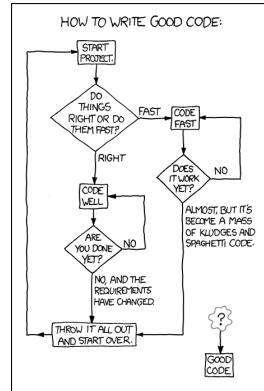
- ▶ inherited magic/spaghetti code
- ▶ data hardcoded into software
- ▶ versioning hell
- ▶ lack of documentation
- ▶ no tests/benchmarks
- ▶ custom data formats
- ▶ hard to modify without breaking
- ▶ hard to install on new machines



# Unsustainable Software

Symptoms of unsustainable software or development practices include:

- ▶ inherited magic/spaghetti code
- ▶ data hardcoded into software
- ▶ versioning hell
- ▶ lack of documentation
- ▶ no tests/benchmarks
- ▶ custom data formats
- ▶ hard to modify without breaking
- ▶ hard to install on new machines



Writing good software is hard. Maintaining it can be even harder...

“**Software sustainability** describes the practices, both technical and non-technical, that allow software to continue to operate as expected in the future. A constant level of effort is required to maintain the software’s operation.”<sup>3</sup>

## Key Considerations

- ▶ Good organisation using version control
- ▶ Ensuring longevity of software, runnable on new hardware/OSes
- ▶ Testing and benchmarking to ensure valid results
- ▶ Documentation - both usage and technical
- ▶ Dissemination/sharing with wider community
- ▶ Community led maintenance effort



---

<sup>3</sup>S.J. Hettrick et al., UK Research Software Survey 2014, DOI:10.5281/zenodo.1183562

Well structured code is:

- ▶ Easy to understand
  - ▷ Consistent code style
  - ▷ Names should explain what things do
  - ▷ Use more, simpler, statements
  - ▷ Comments explain design decisions

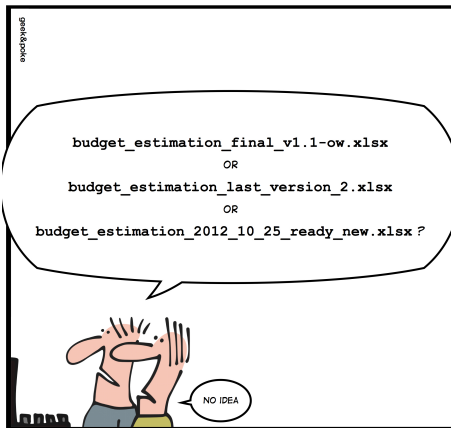
## Well structured code is:

- ▶ Easy to understand
  - ▷ Consistent code style
  - ▷ Names should explain what things do
  - ▷ Use more, simpler, statements
  - ▷ Comments explain design decisions
- ▶ Easy to modify and reuse
  - ▷ Split up source code into logical units
  - ▷ Keep data and code separate
  - ▷ Don't hardcode variables, use a configuration file
  - ▷ Make use of existing language ecosystem

## Well structured code is:

- ▶ Easy to understand
  - ▷ Consistent code style
  - ▷ Names should explain what things do
  - ▷ Use more, simpler, statements
  - ▷ Comments explain design decisions
- ▶ Easy to modify and reuse
  - ▷ Split up source code into logical units
  - ▷ Keep data and code separate
  - ▷ Don't hardcode variables, use a configuration file
  - ▷ Make use of existing language ecosystem
- ▶ Necessary
  - ▷ Use existing libraries where you can
  - ▷ Don't invent your own data formats

# Keeping Code Organised





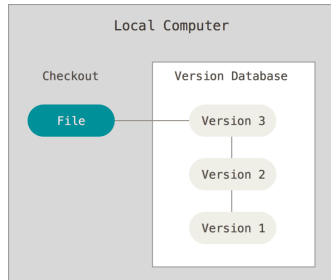
Have you ever

- ▶ Had files with names like “awesomocode\_v4\_final\_final.py”?
- ▶ Made a change to code, then wanted to change it back?
- ▶ Needed to compare two revisions of your code?
- ▶ Had to maintain different versions of your code?
- ▶ Wanted to develop code collaboratively?

Version control systems exist to help with this

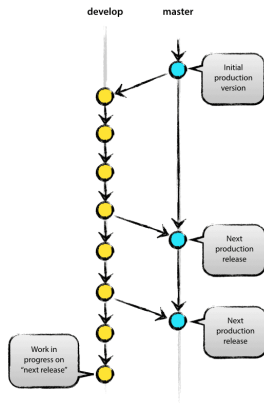
## Version Control Systems (VCS)

- ▶ Keep a full history of changes
- ▶ Easily restore old versions



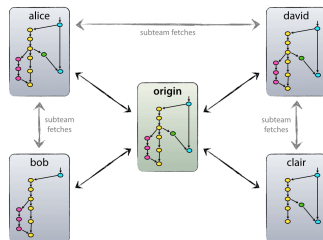
## Version Control Systems (VCS)

- ▶ Keep a full history of changes
- ▶ Easily restore old versions
- ▶ Develop multiple versions together
- ▶ Try things out without consequences



## Version Control Systems (VCS)

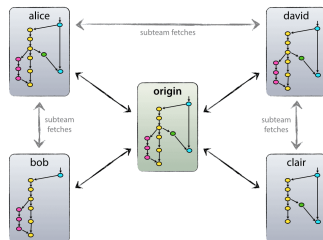
- ▶ Keep a full history of changes
- ▶ Easily restore old versions
- ▶ Develop multiple versions together
- ▶ Try things out without consequences
- ▶ Share code and develop collaboratively



## Version Control Systems (VCS)

- ▶ Keep a full history of changes
- ▶ Easily restore old versions
- ▶ Develop multiple versions together
- ▶ Try things out without consequences
- ▶ Share code and develop collaboratively

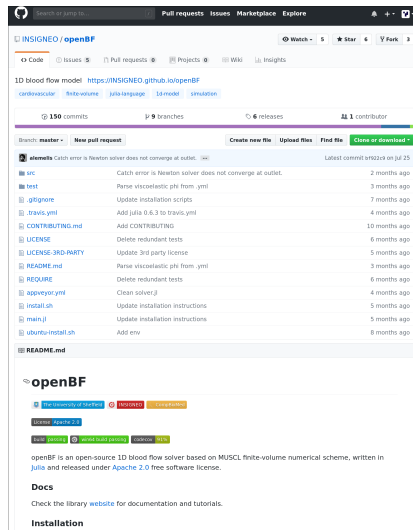
Most commonly used VCS is git

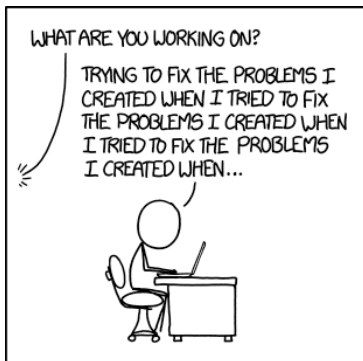


# Keeping Code Organised

## Github

- ▶ Store your git repositories online
- ▶ Public repos for open source software development
- ▶ Free private repos for education and research
- ▶ Impact and engagement tracking
- ▶ Community building tools:
  - ▷ Bug reporting/tracking
  - ▷ Documentation hosting
  - ▷ Project wikis





# Testing and Benchmarking



Testing early and often helps catch mistakes



Testing early and often helps catch mistakes

Ideally test at two different scales:

- ▶ Every function should have accompanying tests (unit tests):
  - ▷ Ensure functions give correct output for correct input
  - ▷ Graceful failures with invalid input
  - ▷ These should be run every time the code is changed

Testing early and often helps catch mistakes

Ideally test at two different scales:

- ▶ Every function should have accompanying tests (unit tests):
  - ▷ Ensure functions give correct output for correct input
  - ▷ Graceful failures with invalid input
  - ▷ These should be run every time the code is changed
- ▶ Test full program behaviour (integration tests):
  - ▷ Identify useful test cases with known results
  - ▷ Test on different machines/architectures
  - ▷ Regression tests: check against previous versions

Testing early and often helps catch mistakes

Ideally test at two different scales:

- ▶ Every function should have accompanying tests (unit tests):
  - ▷ Ensure functions give correct output for correct input
  - ▷ Graceful failures with invalid input
  - ▷ These should be run every time the code is changed
- ▶ Test full program behaviour (integration tests):
  - ▷ Identify useful test cases with known results
  - ▷ Test on different machines/architectures
  - ▷ Regression tests: check against previous versions

Lots of tools to help automate this

## Testing Frameworks

- ▶ Tools to automate running of tests
- ▶ Programmer writes test functions, provides expected output
- ▶ Framework runs all tests and provides report

```
phil@dva ~/philsniftycode $ pytest -v
===== test session starts =====
platform linux -- Python 3.6.6, pytest-3.6.3, py-1.5.4, pluggy-0.6.0 -- /usr/bin/python3.6
cachedir: .pytest_cache
rootdir: /home/phil/philsniftycode, inifile:
plugins: cov-2.5.1
collected 3 items

test_arrays.py::test_build_array PASSED [ 33%]
test_arrays.py::test_copy_array PASSED [ 66%]
test_arrays.py::test_multiply_array FAILED [100%]

===== FAILURES =====
_____ test_multiply_array _____


  def test_multiply_array():
>     raise ValueError("Wrong dimensions in array")
E       ValueError: Wrong dimensions in array

test_arrays.py:14: ValueError
===== 1 failed, 2 passed in 0.03 seconds =====
phil@dva ~/philsniftycode $
```































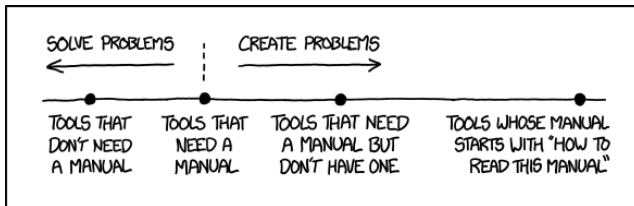
## Continuous Integration

- ▶ Automatically build and test code after changes
- ▶ Test in different environments (linux/windows), compiler versions
- ▶ Free services for open source projects
- ▶ Immediate feedback on bugs/mistakes

INSIGNEO / openBF  build passing

Current Branches Build History Pull Requests More options

 <b>julia7</b>	Remove installation scripts (not needed)	 <b>#85 passed</b>	 11 min 26 sec
 Alessandro Melis		 c37c40d 	 about a month ago
 <b>julia7</b>	Fix compbiomed badge	 <b>#84 passed</b>	 11 min 19 sec
 Alessandro Melis		 9a7105e 	 about a month ago
 <b>julia7</b>	Fix badges	 <b>#83 passed</b>	 11 min 23 sec
 Alessandro Melis		 32b04b0 	 about a month ago
 <b>julia7</b>	direct clone YAML and Codecs	 <b>#81 failed</b>	 6 min 4 sec
 Alessandro Melis		 4bfefae 	 about a month ago



## Getting Users Started

- ▶ Clear installation instructions
- ▶ Concise tutorial (with example data)
- ▶ Explanation of output
- ▶ Troubleshooting?

## User Manual

- ▶ Document all features of the program
- ▶ Details of algorithms and maths used
- ▶ Advanced usage examples
- ▶ Test and benchmarking datasets
- ▶ Known issues, bugs?



## API/Internal Documentation

- ▶ Application Programming Interface - functions for your users to call
- ▶ Document internal functions too
- ▶ Can be autogenerated from code and comments
- ▶ Crucial for future developers (you in 2 years?)

## Documentation Generators

- ▶ Quickly create professional documentation pages or pdf
- ▶ Sphinx and Doxygen are most common
- ▶ Simple human readable format
- ▶ Build api docs from code and comments
- ▶ Use with continuous integration - build docs along with your code and host on github

## Citing

- ▶ Software now citable with DOI
- ▶ Publish on ORDA/Zenodo/Figshare

## Licensing

- ▶ Publically available software should have a license
- ▶ Sustainable software should be open source
- ▶ University may have policy
- ▶ Research Councils may have requirements

## Consumers or Collaborators?

- ▶ Sharing code exposes it to new users and use cases
- ▶ Bugs *will* be found, encourage reporting
- ▶ Open source allows users to engage with development
- ▶ More eyes on code means more issues found and fixed
- ▶ Contribution of new features
- ▶ Users help maintain code for you

Platforms like Github provide all this for minimal effort

# Conclusion: producing good software is hard

We all need to:

- ▶ Structure projects carefully
  - ▶ Use version control to manage development
  - ▶ Test everything!
- 
- ▶ Lots of tools available to help
  - ▶ Building a user community shares the burden
  - ▶ Lots of help out there:
    - ▷ Software Sustainability Institute
    - ▷ University RSE groups
    - ▷ Software Carpentry courses
    - ▷ (Google!)



**Software**  
Sustainability  
Institute



**Research**  
**Software**  
**Engineering**  
**Sheffield.**



## ► Best Practice Guidance

- ▷ RSE@Sheffield <https://rse.shef.ac.uk>
- ▷ SSI: <https://software.ac.uk>
- ▷ Software Carpentry: <https://software-carpentry.org>

## ► Version Control

- ▷ Github: <https://github.com>
- ▷ Bitbucket: <https://bitbucket.org>
- ▷ Git book: <https://git-scm.com/book>

## ► Continuous Integration

- ▷ Travis: <https://travis-ci.org>
- ▷ Appveyor: <https://appveyor.com>