Scientific Software Development: A Pragmatic Approach

Blaise Thompson

University of Wisconsin–Madison

2020-05-07

UW-Madison Chemistry

Chemistry at a glance:

$\sim$ 50 Faculty

$\sim$ 350 Grad. Students

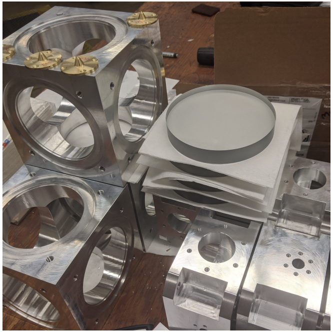
$\sim$ 40 Postdocs

$\sim$ 100 Staff

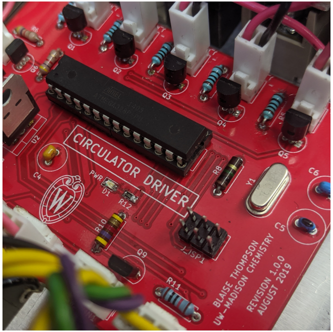
Chemistry Shops

Glass

Machine



Electronics



Blaise Thompson

Who am I?

- Ph.D. Analytical Chemist, 2018 (John C. Wright Group)
- Currently: Instrumentation Technologist
  - design circuits, construct instruments
  - manage shop tools and inventory
  - advise and help researchers
  - software
- No formal training in software development
- Lover of the Python programming language

Scientific Software

Scientists use and develop software for many reasons

- data processing
- driving instrumentation
- modeling

Scientific software projects have a range of scales

- one-off script
- small tool
- large multi-year project

Scientific Software

We are not software developers!
... but somebody has to make the software ...

There are special challenges in scientific software development:

- end-user developers[1,2,3]
- shifting goals[1,2,4,5]
- maintenance[4,5]
- lack of testing[3,5]
- struggles with optimization[5]

1. Segal. "When Software Engineers Met Research Scientists: A Case Study". In: Empirical Software Engineering 10.4 (Oct. 2005), pp. 517–536.
2. Hannay, MacLeod, Singer, Langtangen, Pfahl, Wilson. "How do scientists develop and use scientific software?" In: 2009 ICSE Workshop
3. Joppa, McInerny, Harper, Salido, Takeda *et al.* "Troubling Trends in Scientific Software Use". In: Science 340.6134 (May 2013), pp. 814–815.
4. Carver, Kendall, Squires, Post. "Software Development Environments...: A Series of Case Studies". In: 2007 ICSE Workshop
5. Prabhu, Zhang, Ghosh, August, Huang *et al.* "A survey of the practice of computational science". In: SC '11. ACM Press, 2011.

Scientific Software

…software is profoundly brittle: "small" bugs commonly have unbounded error propagation. …it is rare that a software bug would alter a small proportion of the data by a small amount. More likely, it systematically alters every data point, or occurs in some downstream aggregate step with effectively global consequences. In general, software errors produce outcomes that are inaccurate, not merely imprecise.

F1000Research

F1000Research 2015, 3:303 Last updated: 16 MAY 2019

Check for updates

OPINION ARTICLE

REVISED **Rampant software errors may undermine scientific results [version 2; peer review: 2 approved]**

David A. W. Soergel[1,2]

[1]Department of Computer Science, University of Massachusetts Amherst, Amherst, USA
[2]Current address: Google, Inc., Mountain View, CA, USA

v2 **First published:** 11 Dec 2014, **3**:303 (https://doi.org/10.12688/f1000research.5930.1)
Latest published: 29 Jul 2015, **3**:303 (https://doi.org/10.12688/f1000research.5930.2)

Abstract
The opportunities for both subtle and profound errors in software and data management are boundless, yet they remain surprisingly underappreciated. Here I estimate that any reported scientific result could very well be wrong if data have passed through a computer, and that these errors may remain largely undetected. It is therefore necessary to greatly expand our efforts to validate scientific software and computed results.

Keywords
data management, software error

Open Peer Review

Reviewer Status ✔ ✔

	Invited Reviewers 1	2
REVISED version 2 published 29 Jul 2015	✔ report	✔ report
version 1 published 11 Dec 2014	? report	? report

1 **Titus Brown**, Michigan State University, East Lansing, USA

2 **Daniel S. Katz**, University of Chicago, Chicago, USA

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding author: David A. W. Soergel (david@davidsoergel.com)

Competing interests: No competing interests were disclosed.

Grant information: The author(s) declared that no grants were involved in supporting this work.

How to cite this article: Soergel DAW. **Rampant software errors may undermine scientific results [version 2; peer review: 2 approved]** F1000Research 2015, **3**:303 (https://doi.org/10.12688/f1000research.5930.2)

First published: 11 Dec 2014, **3**:303 (https://doi.org/10.12688/f1000research.5930.1)

Scientific Software

Unlike traditional commercial software developers, but very much like developers in open source projects or startups, scientific programmers usually don’t get their requirements from customers, and their requirements are rarely frozen. In fact, scientists often can’t know what their programs should do next until the current version has produced some results.

Empirical Software Engineering, 10, 517–536, 2005.

When Software Engineers Met Research Scientists: A Case Study

JUDITH SEGAL j.a.segal@open.ac.uk
Department of Computing, Faculty of Mathematics and Computing, The Open University, Milton Keynes, MK7 6AA, UK

Editor: Marvin Zelkowitz

Abstract. This paper describes a case study of software engineers developing a library of software components for a group of research scientists, using a traditional, staged, document-led methodology. The case study reveals two problems with the use of the methodology. The first is that it demands an upfront articulation of requirements, whereas the scientists had experience, and hence expectations, of emergent requirements; the second is that the project documentation does not suffice to construct a shared understanding. Reflecting on our case study, we discuss whether combining agile elements with a traditional methodology might have alleviated these problems. We then argue that the rich picture painted by the case study, and the reflections on methodology that it inspires, has a relevance that reaches beyond the original context of the study.

Keywords: Case study, software engineers, scientific software, agile methodologies, tailoring methodologies.

1. Introduction

This paper describes a case study in which software engineers followed a traditional staged document-led methodology in order to develop a library of instrument-driving software components for a group of research scientists. Two problems with the development are revealed. The first is concerned with requirements: the research scientists are experienced in developing their own software in the laboratory in a highly iterative manner, and having requirements emerge in succeeding iterations. They thus did not appreciate the need to articulate requirements fully and upfront as demanded by a staged methodology, and found this articulation very difficult to do. The second is a problem of communications: using contractual documents (requirement and specification documents) together with formal minuted meetings, did not suffice to construct a fully shared understanding between the scientists and the software engineers.

Case studies can sometimes be viewed with suspicion in the academic empirical software engineering community. Glass et al. (2002), in their review of published research in software engineering, express surprise at the paucity of field/case studies therein. Although Glass et al. did not include the journal of Empirical Software Engineering in their review, a taxonomy of this journal’s publications (Segal et al., 2005), paints a similar picture. But we argue in Segal (2004) and Robinson et al. (2003) that case studies are essential if we are to understand the actual *practice* (as opposed to the theory) of software development, and that such understanding is an essential prerequisite to the primary aim of empirical software engineering, which is to inform practice.

Practices

PLOS | COMPUTATIONAL BIOLOGY

PERSPECTIVE

Good enough practices in scientific computing

Greg Wilson[1]☯*, Jennifer Bryan[2]☯, Karen Cranston[3]☯, Justin Kitzes[4]☯, Lex Nederbragt[5]☯, Tracy K. Teal[6]☯

1 Software Carpentry Foundation, Austin, Texas, United States of America, **2** RStudio and Department of Statistics, University of British Columbia, Vancouver, British Columbia, Canada, **3** Department of Biology, Duke University, Durham, North Carolina, United States of America, **4** Energy and Resources Group, University of California, Berkeley, Berkeley, California, United States of America, **5** Centre for Ecological and Evolutionary Synthesis, University of Oslo, Oslo, Norway, **6** Data Carpentry, Davis, California, United States of America

☯ These authors contributed equally to this work.
* gvwilson@software-carpentry.org

Author summary

Computers are now essential in all branches of science, but most researchers are never taught the equivalent of basic lab skills for research computing. As a result, data can get lost, analyses can take much longer than necessary, and researchers are limited in how effectively they can work with software and data. Computing workflows need to follow the same practices as lab projects and notebooks, with organized data, documented steps

File Format

Choose good file formats!

Start your project by designing your file formats.

- data files
- logs
- configuration files

Use existing file formats where possible.

Prioritize human readability.

Include as much metadata as possible.

File Format

```
# PyCMDS version:       '0.8.0-development'
# system name:  'ps'
# file created:         '2017-10-17T19:00:56.860000+06:00'
# data name:    'diagwmHi'
# data info:    ''
# data origin:  'SCAN'
# queue url:    'https://drive.google.com/open?id=0BzJTClorMBuwZW01WTR6RDhtSGM'
# acquisition url:      'https://drive.google.com/open?id=0BzJTClorMBuwWDhPSEp4Qko4Y2c'
# scan url:     'https://drive.google.com/open?id=0BzJTClorMBuwSEN5aWd2TXExa1E'
# axis names: ['wm'     'w2=w1']
# axis identities: ['wm'        'w2=w1']
# axis units: ['wn'     'wn']
# axis interpolate: [False      False]
# wm points: [ 20800.   20750.          20700.          20650.          20600.          20
# w2=w1 points: [ 1720.         1715.   1710.   1705.   1700.   1695.   1690.   1685.   16
# constant names:       ['w3']
# constant identities:  ['wm-w1-w2']
# channel signed: [False        False   False   False   False   False]
# PCI-6251 shots:       100
# kind: [None   None    None    'hardware'      'hardware'      'hardware'      'hardware'
# tolerance: [None      None    0.01    1.0     1.0     1.0     1.0     1.0     1.0     1.0
# label: [''    ''      'lab'   '1'     'Crystal'       'Amplifier'     'Grating'       'ND
# units: [None  None    's'     'nm'    None    None    None    None    None    None    Non
# name: ['wm_index'     'w2=w1_index'   'time'  'w1'    'w1_Crystal'    'w1_Amplifier'  'w1
0.000000        0.000000        1508284863.268000       5813.953488     -2.725045       1.9
0.000000        1.000000        1508284866.725000       5830.903790     -2.717545       1.9
0.000000        2.000000        1508284870.068000       5847.953216     -2.710045       1.9
0.000000        3.000000        1508284873.457000       5865.102639     -2.702545       1.9
0.000000        4.000000        1508284876.803000       5882.352941     -2.695045       1.9
0.000000        5.000000        1508284880.161000       5899.705015     -2.688295       1.9
```

Blaise Thompson

File Format

```
COLORS  16:29:11, Wed, Jan 17, 2018    COLORS is starting up
Delays_Status.vi      16:29:16, Wed, Jan 17, 2018    Communication with the Newport delay controllers has been initialized on COM4.  Delay commands will be issued through the VISA resource: ASRL4::INSTR
Delays_Status.vi      16:29:19, Wed, Jan 17, 2018    Communication with the Thorlabs delay controller (S/N: 45837036) has been initialized via USB.  Delay commands will be issued through ActiveX.
OPAs_Status.vi  16:29:23, Wed, Jan 17, 2018    OPA0 was unsuccessfully loaded using : ERROR 6033 (TOPAS_OpenDevice.vi (OPA0; <UNDEFINED>))
OPAs_Status.vi  16:29:23, Wed, Jan 17, 2018    OPA1 was successfully loaded using C:-\Users\John\Desktop\COLORS\OPAs\configuration\10743.ini
OPAs_Status.vi  16:29:23, Wed, Jan 17, 2018    OPA2 was successfully loaded using C:-\Users\John\Desktop\COLORS\OPAs\configuration\10742.ini
OPAs_Status.vi  16:29:23, Wed, Jan 17, 2018    OPA3 was unsuccessfully loaded using C:-\Users\John\Desktop\COLORS\OPAs\configuration\OPA3.ini: ERROR 6014 (TOPAS_OpenDevice.vi (OPA3; C:\Users\John\Desktop\COLORS\OPAs\configuration\OPA3.ini))
Delays_Status.vi      16:29:24, Wed, Jan 17, 2018    D1 has been sent to zero
Filter wheels_Status.vi       16:29:24, Wed, Jan 17, 2018    Communication with the filter wheel controller has been initialized on COM18.  Filter wheel commands will be issued through the VISA resource: ASRL18::INSTR
Delays_Status.vi      16:29:27, Wed, Jan 17, 2018    D2 has been sent to zero
Delays_Status.vi      16:29:27, Wed, Jan 17, 2018    Dref has been sent home
Delay_StateCheck (NP).vi      16:29:27, Wed, Jan 17, 2018    ERROR 7750 in Delay_StateCheck (NP).vi (D1) [delta=21.831800 mm] State: READY from MOVING
TOPAS_CalibrateMotor.vi       16:29:30, Wed, Jan 17, 2018    OPA1 motor0 calibration commencing!
TOPAS_CalibrateMotor.vi       16:29:34, Wed, Jan 17, 2018    OPA1 motor1 calibration commencing!
Delay_StateCheck (NP).vi      16:29:35, Wed, Jan 17, 2018    ERROR 7750 in Delay_StateCheck (NP).vi (D1) [delta=4.016830 mm] State: READY from MOVING
```

File Format

File format tips!

- Storing very complex multidimensional data? Consider HDF5.
- Tab characters work best as delimiters.
- TOML and INI are cool.

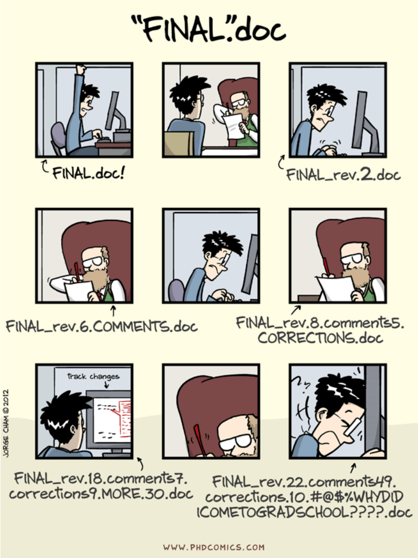
Version Control

Use version control! (probably git)

Software developers use version control to keep track of all of their code changes.

Using version control, you can always return to an earlier version—nothing is lost.

In many cases, the version control system is also the source code backup.



"Not Final" by Jorge Cham - www.phdcomics.com

Version Control

git

https://git-scm.com/

GitHub

https://github.com/

GitLab

https://gitlab.com/

Git is the *ubiquitous* version control system.

- everything is local
- track arbitrary files and folders
- several server options available
  - backup
  - sharing
- can be private if desired

If you do decide to share your code with the world, please consider licensing it.

shops.chem.wisc.edu/training/

Version Control

Katie uses Gaussian computational software in her research. She is exploring a large range of initial conditions using a grid search strategy. Katie uses Git to manage a collection of three or four scripts that she uses to run her simulations and process resulting data. Katie uses the departmental GitLab instance to store her scripts in a private repository. Even though she is the only student working on the project right now, Katie benefits from the version control and backup features as she continues to tweak her script. Katie appreciates the assurance that she can always go back to an earlier version.

Version Control

Louis uses several custom instruments in his daily research. Each of these is a typical analytical/physical chemistry instrument with many components and a large LabVIEW software stack originally written by a long-since-graduated student. Many people rely on these instruments, so it is crucial that their functionality is not interrupted even as Louis improves the software. Louis uses Git to store working versions of the existing LabVIEW code. He then feels confident that he can make edits and improvements without "losing" the old functionality. While he irons out bugs, Louis makes sure that he reverts to the original code so that other users are not interrupted. Louis backs-up the LabVIEW software on the departmental GitLab instance, using a "Group" to ensure that his labmates and advisor also have access.

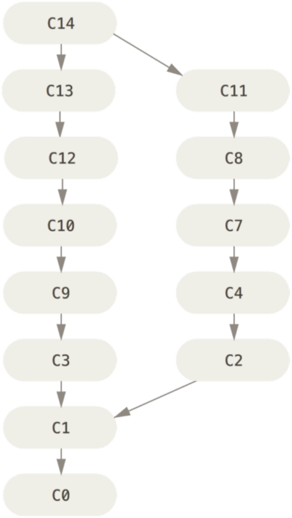


Image adapted from "Pro Git" by Scott Chacon and Ben Straub

Version Control

The Wright Group's research requires them to process large, complex, & multidimensional datasets. In pursuit of this goal, several graduate students spend a significant amount of their time developing a custom data processing library in Python. Due to the scale of this development effort and the number of graduate students working simultaneously on the project, the Wright Group decides to use a branching and pull request workflow to help everyone collaborate. The Wright Group decides to host their code on GitHub, making it publicly available in the hope that other scientists might benefit from and contribute to the library.

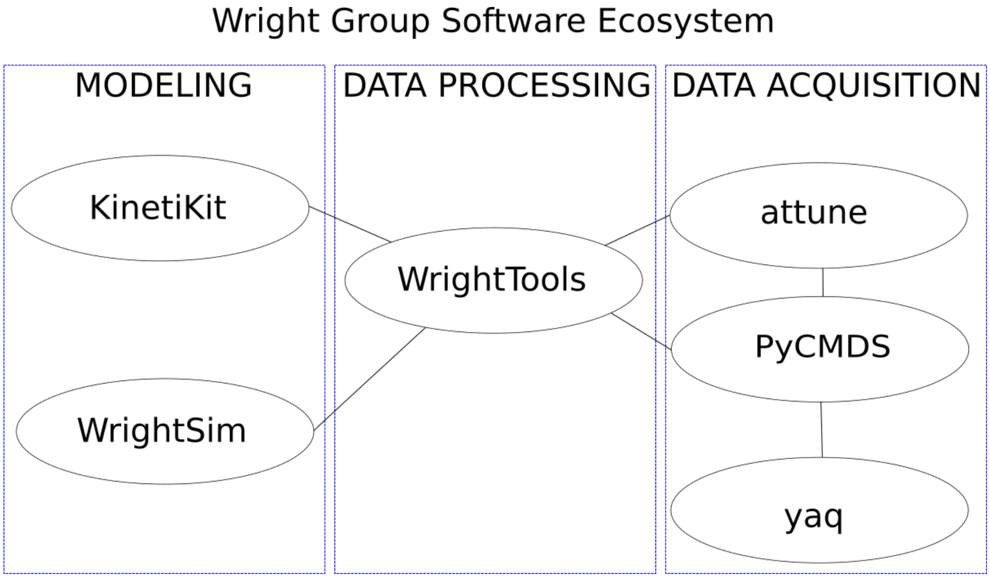
Modularity

Where possible, try to keep software projects small and single purpose.

Focus on interoperability.

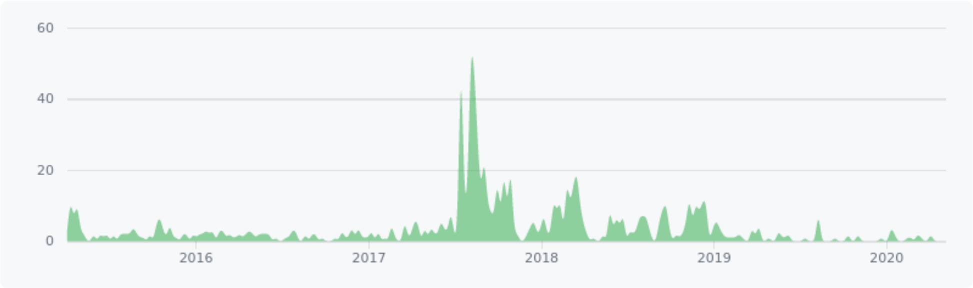
- import your other packages
- shared file formats

Modularity


Wright Group Software Ecosystem
MODELING
DATA PROCESSING
DATA ACQUISITION
KinetiKit
WrightSim
WrightTools
attune
PyCMDS
yaq

Modularity

Software projects *can* be "finished".



Modularity

To help keep different modules interoperable, use tests.

Try to write many small tests that can be run automatically

- when you add a feature
- when you find a bug

Git servers like GitLab and GitHub can automatically run tests for you!

Collaboration

Start by looking for existing projects.

Familiarize yourself with the ecosystem before jumping in.

Don't "reinvent the wheel".

Collaboration

data processing

- yt
- sunpy
- nmrglue
- KOALA
- PyTA
- scikit.ultrafast
- spc
- sncosmo
- scikit-beam

data acquisition

- Exopy
- bluesky
- Instrbuilder
- Lantz
- Qudi
- storm-control
- SFGacquisition
- thorpy
- PyDAQmx

simulation

- COSMOSS
- AutoGAMESS
- pymatgen
- KinetiKit
- CP2K
- GoodVibes
- cctbx
- ChemPy
- phonopy

Collaboration

What if existing projects don't work for you?

Work within small groups

- find common repetitive tasks
- try to "divide and conquer" and share code
- use code review

Optimization

Optimization (making software faster) matters, but *avoid premature optimization*.

Don't get pulled into the trap of trying to make things perfect the first time. Software design is typically a very iterative process, and for good reason. This is particularly true in a scientific context, where goals may evolve during the development process. Write for correctness first, and if it works and is proven useful, consider optimization.

Never optimize blindly—use profiling tools.

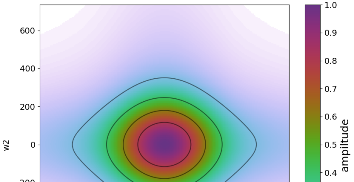
Optimization

PROC. OF THE 17th PYTHON IN SCIENCE CONF. (SCIPY 2018)

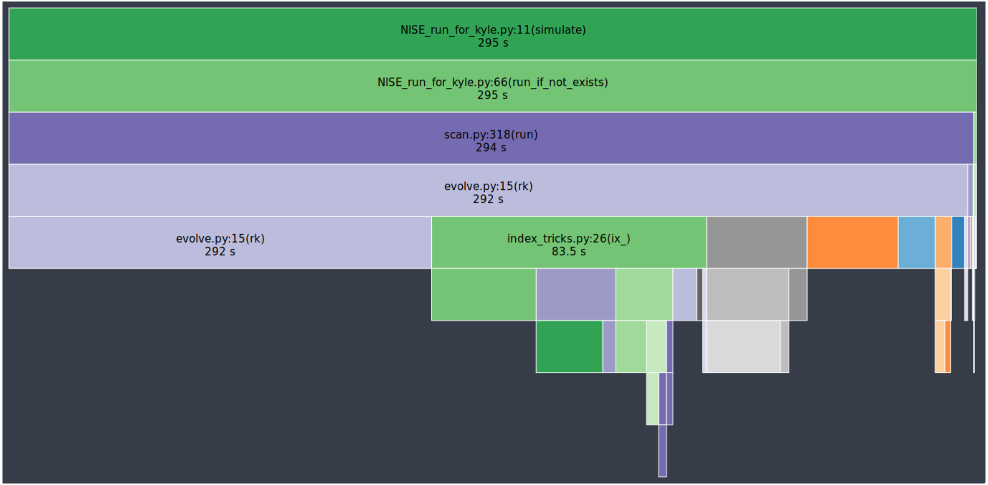
WrightSim: Using PyCUDA to Simulate Multidimensional Spectra

Kyle F Sunden[‡*], Blaise J Thompson[‡], John C Wright[‡]

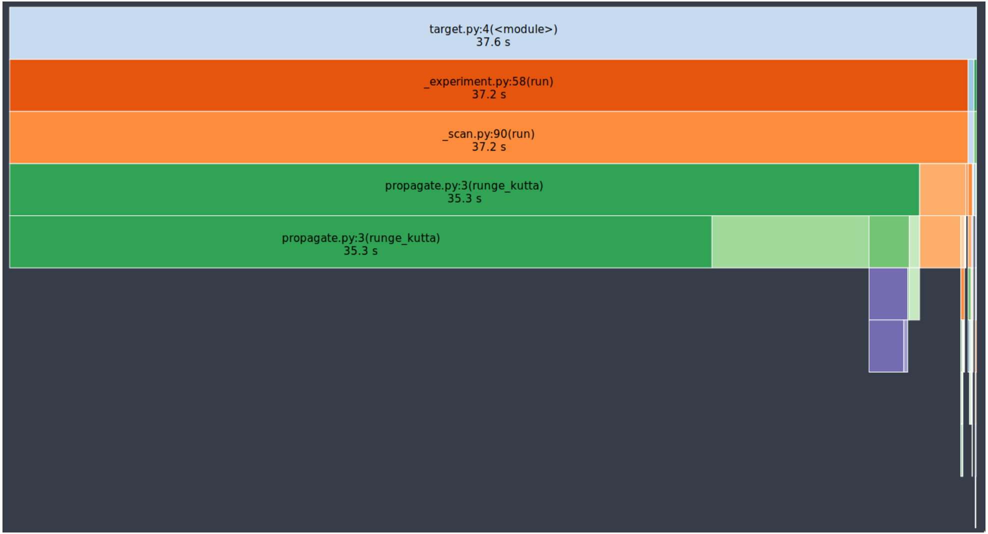
Abstract—Nonlinear multidimensional spectroscopy (MDS) is a powerful experimental technique used to interrogate complex chemical systems. MDS promises to reveal energetics, dynamics, and coupling features of and between the many quantum-mechanical states that these systems contain. In practice, simulation is typically required to connect measured MDS spectra with these microscopic physical phenomena. We present an open-source Python package, `WrightSim`, designed to simulate MDS. Numerical integration is used to evolve the system as it interacts with several electric fields in the course of a multidimensional experiment. This numerical approach allows `WrightSim` to fully account for finite pulse effects that are commonly ignored. `WrightSim` is made up of modules that can be exchanged to accommodate many different experimental setups. Simulations are defined through a Python interface that is designed to be intuitive for experimentalists and theorists alike. We report several algorithmic improvements that make `WrightSim` faster than previous implementations. We demonstrated the effect of parallelizing the simulation, both



Optimization



Optimization



Publish

Publish your scientific software!

- Receive academic credit for your work.
- Communicate about your software to other scientists.
- Provide a citation target.
- Increase reproducibility and decrease effort in your community.

Publish

Distribute your code using standard package managers

- Python Package Index “PyPI”
- MATLAB File Exchange
- The Comprehensive R Archive Network “CRAN”
- VI Package Manager “VIPM”
- Anaconda (multilingual)

Publish

Many Journals:

- Journal of Open Source Software
- Jounral of Open Research Software
- HardwareX
- SoftwareX
- Review of Scientific Instruments

Publish

REVIEW OF SCIENTIFIC INSTRUMENTS **85**, 064104 (2014)

CrossMark
click for updates

KOALA: A program for the processing and decomposition of transient spectra

Michael P. Grubb,[a] Andrew J. Orr-Ewing, and Michael N. R. Ashfold
School of Chemistry, University of Bristol, Cantock's Close, Bristol BS8 1TS, United Kingdom

(Received 31 March 2014; accepted 9 June 2014; published online 26 June 2014)

Extracting meaningful kinetic traces from time-resolved absorption spectra is a non-trivial task, particularly for solution phase spectra where solvent interactions can substantially broaden and shift the transition frequencies. Typically, each spectrum is composed of signal from a number of molecular species (e.g., excited states, intermediate complexes, product species) with overlapping spectral features. Additionally, the profiles of these spectral features may evolve in time (i.e., signal nonlinearity), further complicating the decomposition process. Here, we present a new program for decomposing mixed transient spectra into their individual component spectra and extracting the corresponding kinetic traces: KOALA (Kinetics Observed After Light Absorption). The software combines spectral target analysis with brute-force linear least squares fitting, which is computationally efficient because of the small nonlinear parameter space of most spectral features. Within, we demonstrate the application of KOALA to two sets of experimental transient absorption spectra with multiple mixed spectral components. Although designed for decomposing solution-phase transient absorption data, KOALA may in principle be applied to any time-evolving spectra with multiple components. [http://dx.doi.org/10.1063/1.4884516]

I. INTRODUCTION

A transient absorption experiment captures the time-resolved dynamics of a chemical process from the time-evolution of its absorption spectrum. Generally, a pump laser

such as Principal Component Analysis, Independent Component Analysis,[6] and Multivariate Curve Resolution.[7] The drawback of blind source separation methods is that the underdetermined nature of the decomposition problem often re-

Publish

The Journal of Open Source Software

WrightTools: a Python package for multidimensional spectroscopy

Blaise J. Thompson[1], **Kyle F. Sunden**[1], **Darien J. Morrow**[1], **Daniel D. Kohler**[1], **and John C. Wright**[1]

1 University of Wisconsin–Madison

DOI: 10.21105/joss.01141

Software

- Review
- Repository
- Archive

Submitted: 16 December 2018
Published: 17 January 2019

Introduction

"Multidimensional spectroscopy" (MDS) is a family of analytical techniques that record the response of a material to multiple stimuli—typically multiple ultrafast pulses of light. This approach has several unique capabilities;

Publish

HTML ABSTRACT + LINKS

REVIEW OF SCIENTIFIC INSTRUMENTS VOLUME 75, NUMBER 1 JANUARY 2004

LabView virtual instrument for automatic plasma diagnostic

J. Ballesteros,[a] J. I. Fernández Palop, M. A. Hernández, R. Morales Crespo, and S. Borrego del Pino
Departamento de Física, Campus Universitario de Rabanales, Edificio C2, Universidad de Córdoba, 14071 Córdoba, Spain

(Received 19 June 2003; accepted 27 October 2003)

This article presents a LabView virtual instrument (VI) that automatically measures the $I-V$ plasma probe characteristic and obtains the electron energy distribution function (EEDF) in plasmas. The VI determines several parameters characterizing the plasma using different methods to verify the validity of the results. The program controls some parameters associated with color coded warnings to verify the fidelity of the measured data and their later numerical treatment. The measurement process and data treatment are very fast, about 0.5 s, so that temporal evolutions of the EEDF can be scanned, to analyze the drift of the plasma. Finally, the program is easily portable since it is developed in the LabView environment, so it can be adapted to any platform using common laboratory instruments. © *2004 American Institute of Physics.* [DOI: 10.1063/1.1634356]

I. INTRODUCTION

The plasma diagnostic techniques which use the $I-V$ characteristic of a Langmuir probe immersed in it, are classic and broadly used. Applying the adequate technique for each kind of plasma, local information about the parameters characterizing the plasma can be obtained, e.g., the density and temperature of the species composing it, the plasma poten-

whole process can be carried out in a single step by performing a convolution with the following function:[14]

$$g_n(x)=\sum_{k=1}^{n}\binom{n}{k}(-1)^{k+1}\frac{\alpha}{\sqrt{\pi k}}e^{-\alpha^2x^2/k}, \qquad (2)$$

$\sigma=1/\alpha\sqrt{2}$ being the standard deviation of the Gaussian distribution function and n the number of iterations.

Publish

.. if your code is good enough to do the job, then it is good enough to release — and releasing it will help your research and your field.

WORLD VIEW *A personal take on events*

WWW.SERENAATKINS.COM

Publish your computer code: it is good enough

Freely provided working code — whatever its quality — improves programming and enables others to engage with your research, says **Nick Barnes**.

I am a professional software engineer and I want to share a trade secret with scientists: most professional computer software isn't very good. The code inside your laptop, television, phone or car is often badly documented, inconsistent and poorly tested.

Why does this matter to science? Because to turn raw data into published research papers often requires a little programming, which means that most scientists write software. And you scientists generally think the code you write is poor. It doesn't contain good comments, have sensible variable names or proper indentation. It breaks if you introduce badly formatted data, and you need to edit the output by hand to get the columns to line up. It includes a routine written by a graduate student which you never completely understood, and so on. Sound familiar? Well, those things don't matter.

That the code is a little raw is one of the main reasons scientists give for not sharing it with others. Yet, software in all trades is written to be good enough for the job intended. So if your code is good enough to do the job, then it is good enough to release — and releasing it will help your research and your field. At the Climate Code Foundation, we encourage scientists to publish their software. Our experience shows why this is important and how researchers in all fields can benefit.

Programs written by scientists may be small scripts to draw charts and calculate correlations, trends and significance, larger routines to process and filter data in more complex ways, or telemetry software to control or acquire data from lab or field equipment. Often they are an awkward mix of these different parts, glued together with piecemeal scripts. What they have in common is that, after a paper's publication, they often languish in an obscure folder or are simply deleted. Although the paper may include a brief mathematical description of the processing algorithm, it is rare for science software to be published or even reliably preserved.

Last year's global fuss over the release of climate-science e-mails from the University of East Anglia (UEA) in Norwich, UK, highlighted the issue, and led the official inquiry to call for scientists to publish code. My efforts pre-date the UEA incident and grew from work in 2008 based on software used by NASA to report global temperatures. Released on its website in 2007, the NASA code was messy and proved difficult for critics to run on their own computers. Most did not seem to try very hard, and nonsense was written about fraud

them and now intends to replace its original software with ours.

So, openness improved both the code used by the scientists and the ability of the public to engage with their work. This is to be expected. Other scientific methods improve through peer review. The open-source movement has led to rapid improvements within the software industry. But science source code, not exposed to scrutiny, cannot benefit in this way.

NO EXCUSES

If scientists stand to gain, why do you not publish your code? I have already discussed misplaced concern about quality. Here are my responses to some other common excuses.

It is not common practice. As explained above, this must change in climate science and should do so across all fields. Some disciplines, such as bioinformatics, are already changing.

People will pick holes and demand support and bug fixes. Publishing code may see you accused of sloppiness. Not publishing can draw allegations of fraud. Which is worse? Nobody is entitled to demand technical support for freely provided code: if the feedback is unhelpful, ignore it.

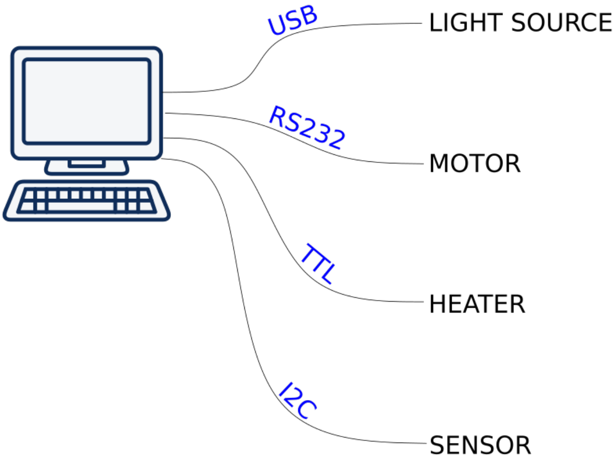
The code is valuable intellectual property that belongs to my institution. Really, that little MATLAB routine to calculate a two-part fit is worth money? Frankly, I doubt it. Some code may have long-term commercial potential, but almost all the value lies in your expertise. My industry has a name for code not backed by skilled experts: abandonware. Institutions should support publishing; those who refuse are blocking progress.

It is too much work to polish the code. For scientists, the word publication is totemic, and signifies perfectionism. But your papers need not include meticulous pages of Fortran; the original code can be published as supplementary information, available from an institutional or journal website.

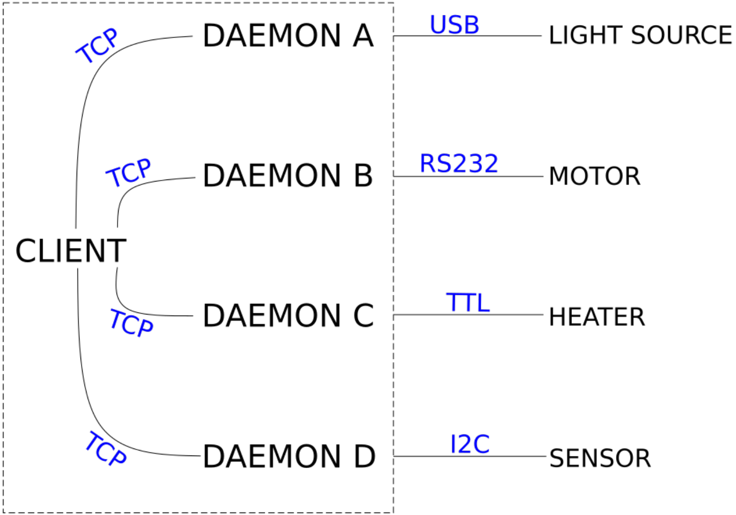
I accept that the necessary and inevitable change I call for cannot be made by scientists alone. Governments, agencies and funding bodies have all called for transparency. To make it happen, they have to be prepared to make the necessary policy changes, and to pay for training, workshops and initiatives. But the most important change must come in the attitude of scientists. If you are still hesitant about releasing your code, then ask yourself this question: does it perform the algorithm you describe in your paper? If it does, your audience will accept it, and maybe feel happier with its own efforts to write programs. If not, well,

NOBODY IS ENTITLED TO DEMAND TECHNICAL SUPPORT FOR FREELY PROVIDED CODE: IF THE FEEDBACK IS UNHELPFUL, IGNORE IT.

yaq



yaq


CLIENT
TCP
TCP
TCP
TCP
DAEMON A
DAEMON B
DAEMON C
DAEMON D
USB
RS232
TTL
I2C
LIGHT SOURCE
MOTOR
HEATER
SENSOR

yaq

yaq daemons

- develop each daemon, client separately
- can be implemented in any language
- more reusable
- less fragile

yaq

where possible, yaq attempts to enforce consistency between different interfaces

is-sensor

- measure
- get-measured
- stop-looping
- get-channel-names
- get-channel-shapes
- get-channel-units

has-position

- get-destination
- get-units
- get-position
- set-position
- set-relative

is-homeable

- home

yaq

Tanner is building a continuous flow reactor to allow him to do kinetics studies on novel polymer chemistries. He builds his reactor using a few commercial available pumps, valves, and sensors which are "lying around the lab". Tanner is currently the only scientist working on this project, and the reactor is under heavy development as he continues to refine his experimental procedures. Rather than creating a monolithic graphical user interface, Tanner uses yaq to interface with his hardware and writes simple $\sim$ 50 line Python scripts to drive his reactions. As Tanner continues to change his reactor, he can easily make a new script that ensures his valves and pumps fire in the appropriate pattern.

yaq

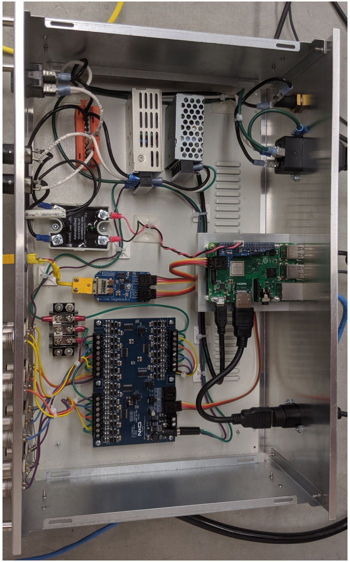
Louis and Kurtis have been asked to build a pH-stat out of a pair of syringe pumps and a pH meter. They would like to get the programming out of the way as quickly as possible. Louis and Kurtis separately develop yaq daemons to interface with the pH meter and syringe pumps respectively. With their daemons working well, they meet back in the wetlab and quickly fine-tune a simple client.

yaq

The Wright Group relies on the flexibility of their laser systems to accomplish a wide variety of experimental procedures. Graduate students in this group frequently find themselves switching out hardware on the laser table. Because they use yaq to interface with their hardware, the Wright Group can write a generic client which capitalizes on the shared traits system. As long as their client is familiar with that particular “class” of hardware, Wright Group graduate students can add and remove instrument components at will. The graduate students find that they can run both of their laser tables using the same generic client.

yaq

Chase is building a pressurized reactor including a custom isothermal block and sensors. This reactor will be installed into a wetlab environment, so it's hard to find the place to put a computer, monitor, keyboard, and mouse. Instead, Chase uses a Rasperry Pi that's connected to the network. He implements his interfaces to the sensors and heaters in yaq, and controls them remotely from a laptop on the counter or from the comfort of his office.



Thank You

Thank you for your attention
... any questions?

Contact me:

- bthompson@chem.wisc.edu
- https://shops.chem.wisc.edu
- https://yaq.fyi
- https://wright.tools

Dont forget: *use version control today!*