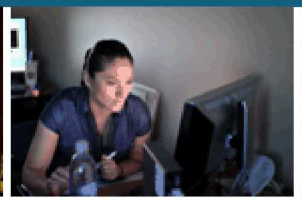
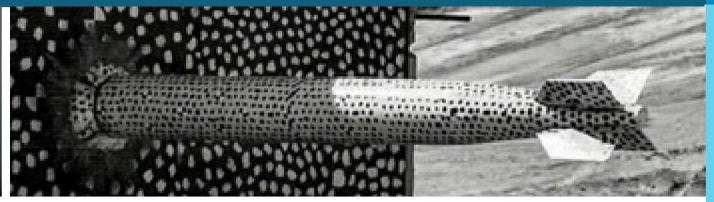
SAND2018-12452PE

Some Perspectives on Testing and Continuous Integration for Open Source Software

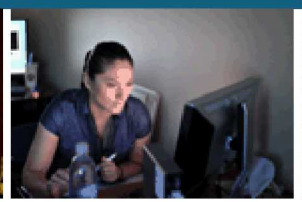


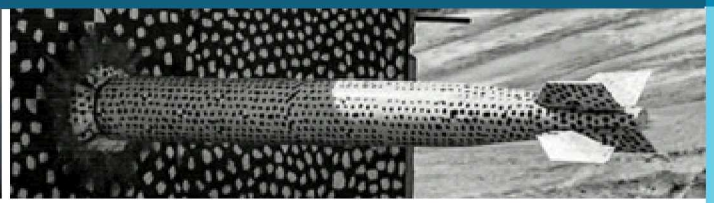


William Hart and John Siirola
Sandia National Laboratories
wehart@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Some Perspectives on Software Quality Management for Open Source Software





William Hart and John Siirola
Sandia National Laboratories
wehart@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Why do we care about software quality in research codes?

For ourselves
- Reputation
- Research quality
- Communication of ideas

For our jobs
- These are capabilities we can use for
  - National security missions
  - Commercial applications
  - Education

On the Impact of Software Testing - Three Examples

Optimization Methods for AutoDock

- My thesis work analyzed GA hybrids with local search
- My analysis for why these methods worked was flawed because it ignored scaling issues
- **Lesson:** Testing can help you better understand how your code works

Tracking Down a bug in Acro

- During Acro development, it was common to have unresolved test failures between releases
- These test failures masked a flaw that was introduced but not realized until months later
- But the online test history proved invaluable for tracking down when this was introduced
- **Lesson:** Testing archives provide a rich context for identifying subtle bugs

Enabling Major Changes

- Pyomo recently replaced its expression tree logic
- High test coverage provided confidence that the software was working as expected
- **Lesson:** Strong tests can catalyze major software changes

What Is Working Well?

What Is Working Well?

Cloud hosting

Repository management

Online documentation generation

Automated build, distribution and deployment

Automated testing and code coverage analysis

Cloud Hosting

Repository hosting
- GitHub, GitLab, SourceForge, BitBucket, …

Testing
- TravisCI, Appveyor, GitLab, Jenkins, circleci, …

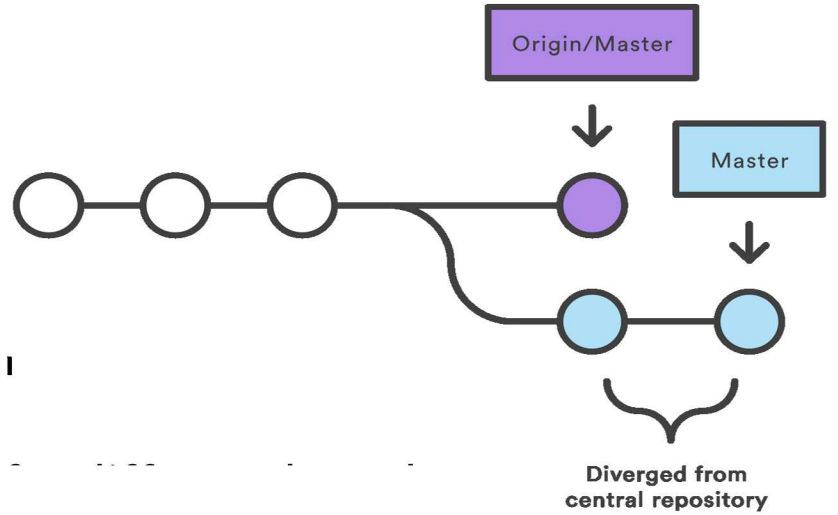
Package management
- PyPI, conda-forge, Julia/GitHub, …

Mostly free to academics and open-source projects

Repository Management

Distributed repository management is much more flexible than previous paradigms

- E.g. using CVS or Subversion



Example: Git Workflows

- E.g. see https://www.atlassian.com/git/tutorials/compa
  - *Centralized* - development occurs on the master branch
  - *Feature branching* - major features developed on sep
  - *GitFlow* - feature branching will well-defin
  - *Forking workflow* - developers work on forks of the main repository
- **Impact**: teams can adopt workflows that are better tailored to their needs and resources

Example: Pull Requests

- A method of submitting contributions to a project using a distributed VCS like git
- Can automate application of tests, coverage analysis, static code analysis, etc
- **Impact**: can ensure stability of the master branch (or other stable branches)

Online Documentation

Online documentation used to be a link to a PS/PDF file

Markup languages are commonly supported in various platforms

- E.g. GitHub wiki pages
- E.g. see https://en.wikipedia.org/wiki/List_of_document_markup_languages

Sophisticated HTML/PDF documents can be generated automatically

- Autodoc
- ReadTheDocs, readme.io
- apiary.io

Automated build, distribution and deployment

Cloud build/test environments

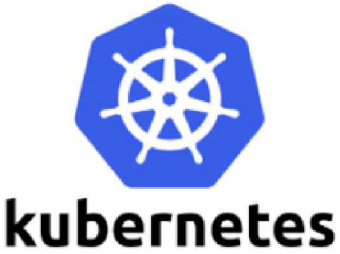
- TravisCI, CircleCI, Appveyor, Jenkins, CodeShip, Bamboo, TeamCity, …
- Hooks for up-loading distributions

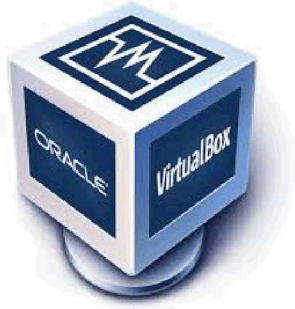
Conda/Conda-Forge

- Build + distribution management

Containers/Virtualization

- Docker
- Kubernetes
- VirtualBox





Azure Pipelines

Automated testing and code coverage analysis

Cloud build/test environments

- TravisCI, CircleCI, Appveyor, …

Travis CI

AppVeyor

circleci

Jenkins

CODESHIP

Bamboo

TeamCity

Flexible testing tools

- Java: jUnit, …
- Python: unittest, nose
- C++: CppUnit, CxxTest, Ctest, Boost, Aeryn, NanoCppUnit, GoogleTest, unittest-cpp, onqtam/doctest, philsquared/Catch

Flexible code coverage analysis

- Python: coverage.py
- C++: lcov, gcovr

Build Services and Continuous Integration

What *is* Continuous Integration / Continuous Deployment?

- Common (best) practices [1]:
  - Maintain a code repository
  - Automate the build
  - Make the build self-testing
  - Everyone commits to the baseline every day
  - Every commit (to the baseline) should be built
  - Keep the build fast
  - Test in a clone of the production environment
  - Make it easy to get the latest deliverables (builds)
  - Everyone can see the results of the latest build
  - Automate deployment

- But does COIN-OR really want (need) CI/CD?

[1] From https://en.wikipedia.org/wiki/Continuous_integration

Build Services and Continuous Integration

What *is* Continuous Integration / Continuous Deployment?

- Common (best) practices [1]:
  - √ Maintain a code repository
  - √ Automate the build
  - √ Make the build self-testing
  - ? Everyone commits to the baseline every day
  - ? Every commit (to the baseline) should be built
  - ? Keep the build fast
  - ? Test in a clone of the production environment
  - √ Make it easy to get the latest deliverables (builds)
  - ? Everyone can see the results of the latest build
  - √ Automate deployment

- But does COIN-OR really want (need) CI/CD?

[1] From https://en.wikipedia.org/wiki/Continuous_integration

Do we really need the *integration* in CI?

Everyone commits to the baseline every day

CI/CD is particularly targeted toward large(r) teams focused on rapid development

- Coordinate team members through frequent integration with master
- Avoid large (and painful) merge conflicts

COIN-OR's situation is different

- (Most) projects are actually small teams
- COIN-OR development is no one's "day job"
  - Development velocity is slower
  - Velocity varies dramatically across a team and in time
- New features may take weeks to months to mature
  - Long-running feature branches

Do we have the resources to pull off CI/CD?

Every commit (to the baseline) should be built
Keep the build fast
Test in a clone of the production environment

COIN-OR is targeting a *huge* production environment

- Windows, Linux (Ubuntu, RHEL, …), OSX
- Python (2.7, 3.4, 3.5, 3.6, 3.7…)
- Julia (0.7, 1.0, …)
- Compilers (gcc, icc, msvs), Matlab, R, …

(Free) CI/CD providers either limit concurrency, or runtime, or both

- E.g., Appveyor (Windows) runs jobs sequentially
- E.g., Travis limits concurrency to ~5-6 jobs

Build pipelines generally not supported (in free offerings)

- Azure Pipelines is an interesting exception

Can third-party services meet all our needs?

Test in a clone of the production environment
Everyone can see the results of the latest build

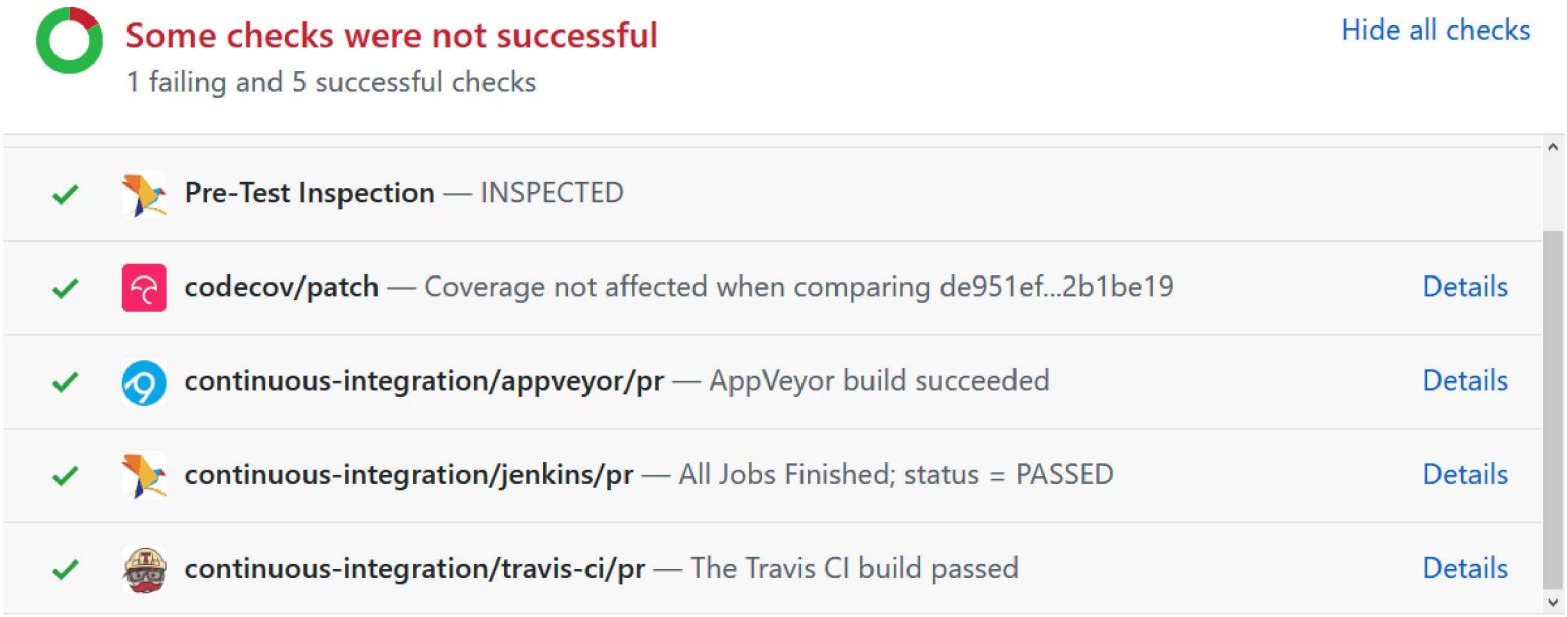
We regularly need non-public libraries, tools, applications that are not available on third-party public build platforms

- Matlab, HSL, Cplex/Gurobi/Xpress, (GAMS)

Private build services can support proprietary tools, but…

- Securing the server is a near-fulltime job
  - E.g., Pyomo leverages corporate infrastructure (firewalls, access control, system administration, application distribution infrastructure, Jenkins instance)
- Getting developers access to the build results is … difficult
- Building PR's from third parties represents a security risk
  - But we have a solution…

Private (or non-free) CI services can provide complex build workfllows



Testing and CI/CD

Testing is an absolute necessity, but not all tests are created equal…

- Unit testing
- Integration testing
- Regression testing
- Performance testing

Some anecdotes

- Good unit test frameworks exist (e.g., cxxtest, unittest, pytest, …)
- Unit testing must be integrated into the code design
  - "Unit" testing 300-line functions is almost impossible
- Unit testing is *hard* and *time consuming*
  - I usually see 2:1 lines of unit test code to lines of production code
- 100% code coverage is *necessary* but not *sufficient*
  - Just *yesterday* I fixed a bug in a subsection of new Pyomo code that already had 100% coverage
  - *Running* the code is not the same as *testing* the code
    - Regression tests are very good at exercising large amounts of code, but are weak tests
    - Unit tests without assertions are barely tests
  - Code coverage != *Branch coverage*
  - A goal: 100% coverage of a unit by *only running that unit's tests*
    - …but this kind of testing / reporting is not supported by any automation system I know of

What could be better?

What could be better?

Performance testing

Test integration across projects

Large-scale testing

Effective software project management

Performance Testing

Challenge: Tracking code performance during development

Cloud hosting platforms are often not suitable for this

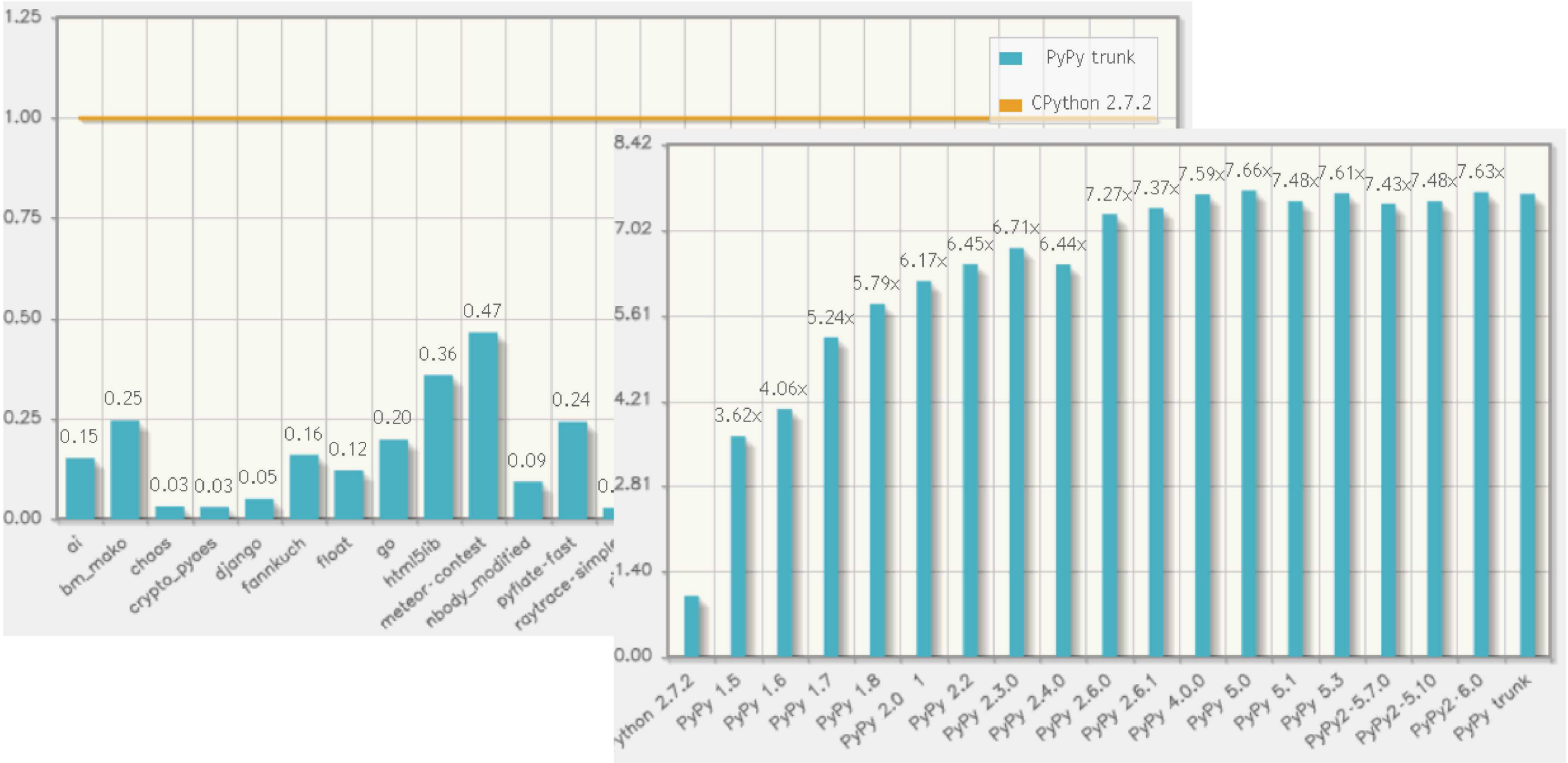
Few exemplars for benchmarking

- PyPy - http://speed.pypy.org/
- OS/Component benchmarks - https://openbenchmarking.org/

Software Performance Testing

- Test Types: Load, Stress, Soak, Spike, Breakpoint, Configuration, Isolation

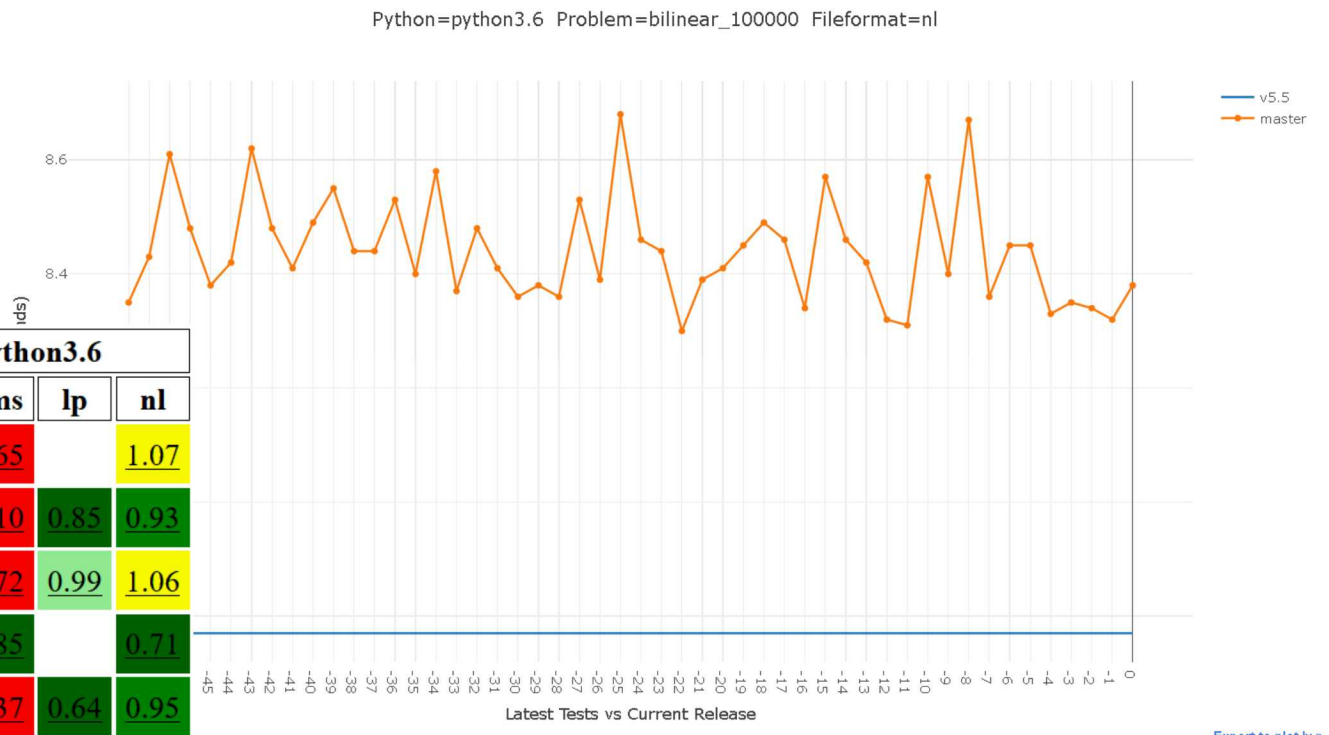
Example: PyPy



Performance Testing with Pyomo

Goal: Track benchmark performance relative to last release

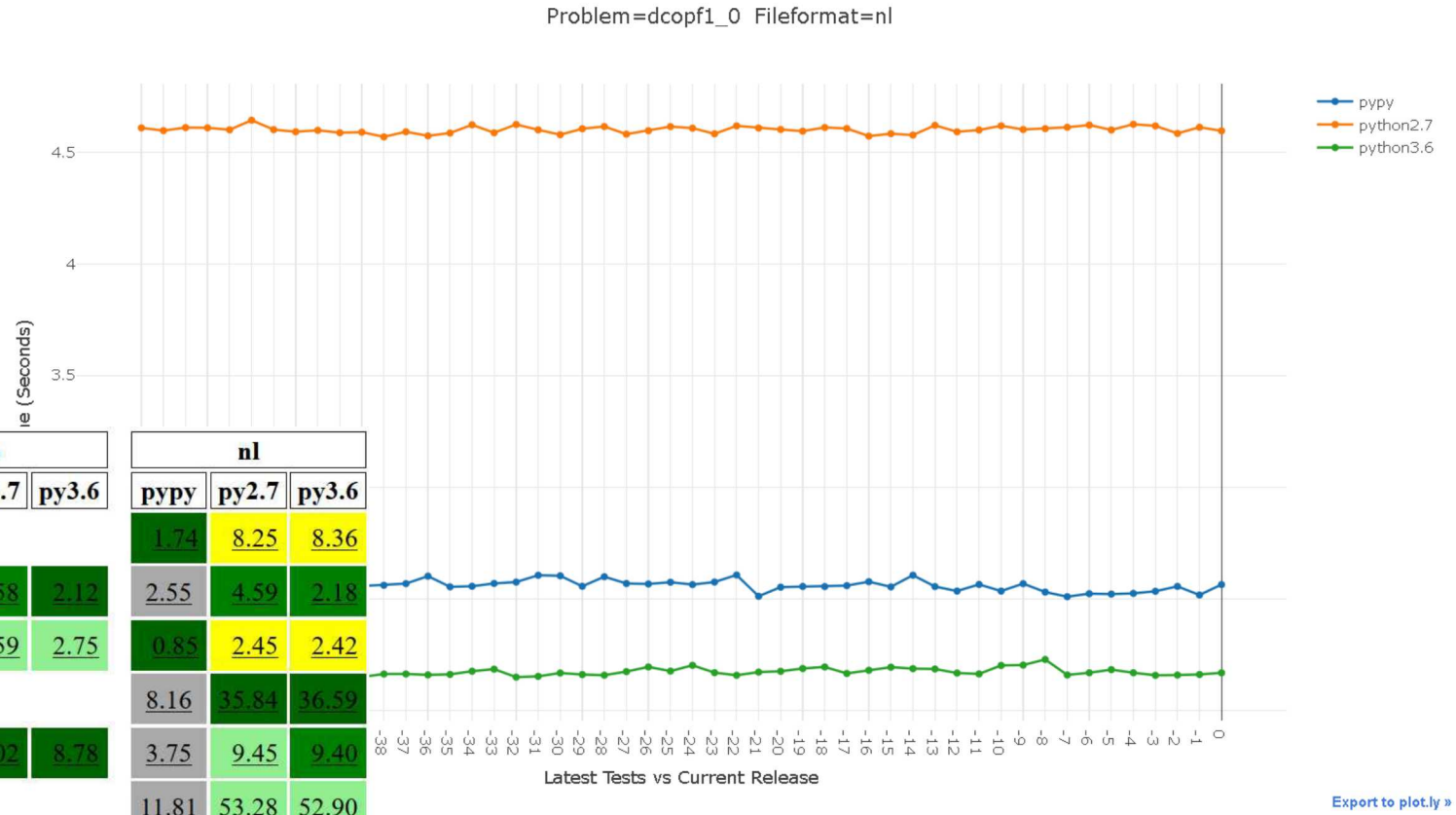
	pypy				python2.7				python3.6			
Problem	**bar**	**gms**	**lp**	**nl**	**bar**	**gms**	**lp**	**nl**	**bar**	**gms**	**lp**	**nl**
bilinear_100000	NA	0.97		0.87	NA	1.40		1.08	NA	1.65		1.07
dcopf1_0	0.16	0.21	NA	NA	0.14	1.00	0.90	0.93	0.06	1.10	0.85	0.93
diag_100000	NA	1.08	0.98	0.89	NA	1.60	1.02	1.08	NA	1.72	0.99	1.06
jump_clnlbeam_50000	NA	NA		NA	NA	0.73		0.68	NA	0.85		0.71
jump_facility_25	NA	NA	NA	NA	NA	1.27	0.63	0.96	NA	1.37	0.64	0.95
jump_lqcp_500	NA	NA		NA	NA	1.61		1.04	NA	1.84		1.04
jump_opf_6620	NA	NA		NA	NA	1.29		1.21	NA	1.43		1.17
pmedian_8	NA	NA	0.94	0.94	NA	1.39	0.98	0.99	NA	1.53	0.94	0.99
stochpdegas1_0	NA	NA		NA	0.04	NA		0.99	0.05	NA		1.02
uc1_0	NA	NA	NA	NA	NA	1.33	0.98	0.99	NA	1.45	0.96	0.97



Performance Testing with Pyomo

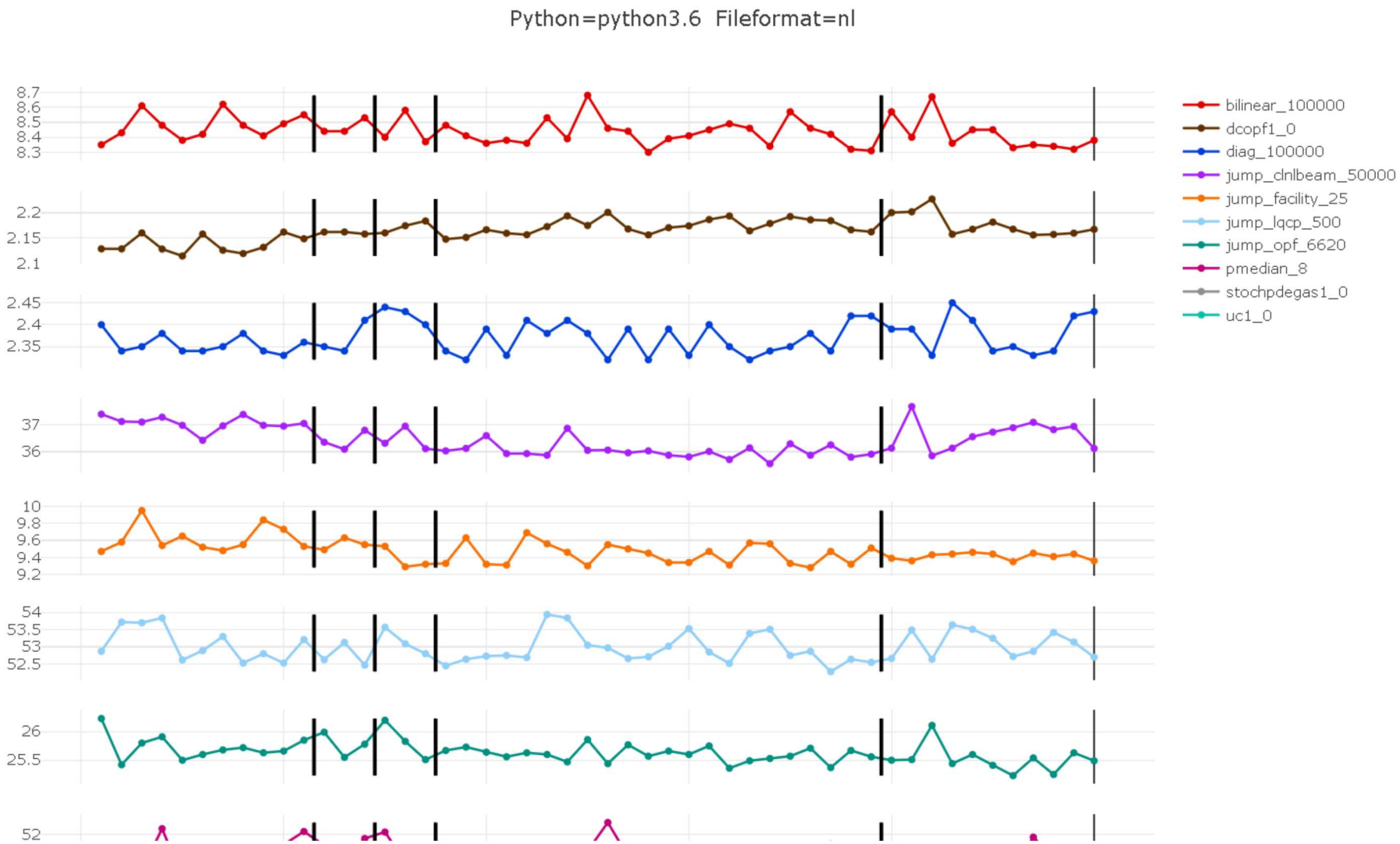
Goal: Track relative performance of different versions of Python

	bar			gms			lp			nl		
Problem	**pypy**	**py2.7**	**py3.6**	**pypy**	**py2.7**	**py3.6**	**pypy**	**py2.7**	**py3.6**	**pypy**	**py2.7**	**py3.6**
bilinear_100000	1.57	7.72	8.32	1.25	8.75	9.68				1.74	8.25	8.36
dcopf1_0	2.50	4.46	2.09	2.69	4.57	2.24	2.65	4.58	2.12	2.55	4.59	2.18
diag_100000	0.73	2.97	3.32	0.70	4.02	4.29	1.13	2.59	2.75	0.85	2.45	2.42
jump_clnlbeam_50000	6.15	31.62	34.41	5.66	32.59	35.99				8.16	35.84	36.59
jump_facility_25	2.82	9.08	9.11	2.95	9.23	9.95	3.70	8.02	8.78	3.75	9.45	9.40
jump_lqcp_500	10.91	60.41	65.14	10.74	73.24	78.75				11.81	53.28	52.90
jump_opf_6620	6.35	18.19	19.86	6.56	21.86	23.34				8.90	25.33	25.41
pmedian_8	12.86	54.43	58.31	10.11	56.51	60.65	14.88	46.81	51.02	14.21	51.61	51.33
stochpdegas1_0	3.12	1.57	1.67	3.19	1.61	1.80				3.64	1.76	1.83
uc1_0	17.16	67.21	69.44	15.42	70.43	75.23	19.86	60.80	64.75	18.77	64.94	64.66



Performance Testing with Pyomo

Goal: Identify points where performance changed



Test Integration Across Projects

Challenge: quality management across interdependent libraries

Example: pyomocontrib_simplemodel

- A simple, PuLP-like modeling environment built using Pyomo
- Commits to Pyomo branches
  - Trigger tests for Pyomo
- Commits to Pyomo master branch
  - Trigger tests for the master branch of pyomo_simplemodel

Problems:

- Setting up this type of testing is not straightforward
- We probably want to test against specific versions and specific branches, which gets complex
- How do we collect/summarize test results for a large project like COIN-OR?
- Different developer groups might want to focus on different collections of stable sub-projects

Large-Scale Testing

Challenge: running tests in a timely manner

Example: pyomo

- Currently tests 11 combinations on TravisCI (Python versions x Subtests) and tests 4 combinations on Appveyor
  - Test runtime and concurrency limitations prevent more Appveyor tests!

Example: pyomo benchmarks

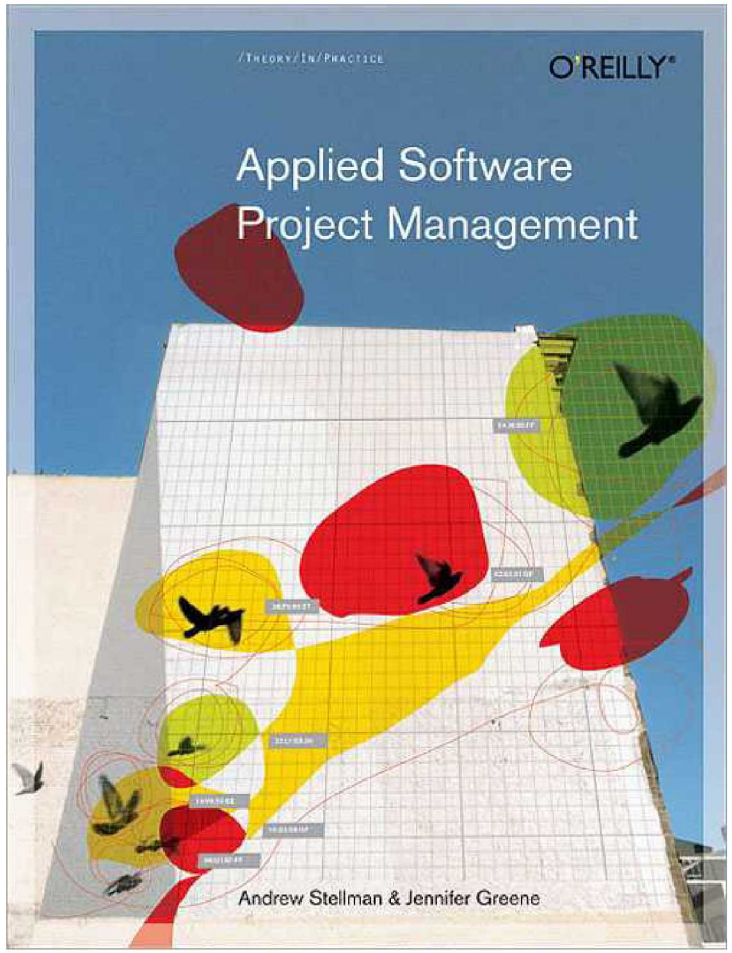
- Currently run nightly, testing ~200 combinations
  - (Python versions x Problems x IO x Configurations)
  - Benchmark tests take 10+ hours each day on a dedicated server
- Cannot run limited performance tests in an agile manner
- Cannot easily integrate testing results across multiple days
  - E.g. since the codes doesn't always change…

Effective Software Project Management

Challenge: managing human capital

Key Project Management Activities

- Project planning
- Estimation
- Project schedules
- Reviews
- Software requirements
- Design and programming
- Software testing
- Managing change
- Project management



Effective Software Project Management

Challenge: managing human capital

Key Project Management Activities

- Project planning
- Estimation
- Project schedules
- Reviews
- Software requirements
- Design and programming
- Software testing
- Managing change
- Project management

Most of these activities are intrinsically labor-intensive

Effective Software Project Management

Challenge: managing human capital

Key Project Management Activities

- Project planning
- Estimation
- Project schedules
- Reviews
- Software requirements
- Design and programming
- Software testing
- Managing change
- Project management

Most of these activities are intrinsically labor-intensive

Thus, better tools aren't the **solution** to effective project management

Thus, effective management of our human capital is key

- Time to lead projects
- Time for design/implementation/reviews
- Time for documentation
- Time to iterate (re-design/re-implement/…)

Effective Software Project Management

Challenge: new research vs. re-usable components

Observation:

- We need both!
- COIN-OR and related initiatives are the embodiment of academic and government research
- And our own research needs to rely on a stable foundation of existing capabilities

How do we manage change that is highly intrusive?

- E.g. Pyomo replaced its expression system this year (!)
- E.g. Direct solver interfaces for Pyomo (see Carl Laird's talk)
- E.g. Rethinking the design of COIN-OR solver interfaces

Final Observation

There are fundamentally different challenges for open source research codes like COIN-OR.

There is high interest in developing research codes, but it is hard to recruit people to maintain these codes.

- Because the research is done!

What is good enough for research is often not good enough for practice

- E.g. grad-ware

Research does not have a well-defined target

- The requirements are likely to change
- A seemingly good design today can be easily criticized tomorrow
- It is hard to estimate the effort needed to develop new codes