

Software Engineering for Research Software (SE4RS'23)

[Attendees \(Name, Affiliation, Email\)](#)

[Paper Session](#)

[Large Group Discussion](#)

[Discussion/Speed Blogging Topics](#)

[Topic 1: How to get researchers to use good practices?](#)

[Topic 2: Creating/sustaining RSE teams](#)

[Topic 3: xxxx](#)

[Topic 4: xxxx](#)

[Topic 5: xxxx](#)

[Topic 6: xxxx](#)

[Topic 7: xxxx](#)

[Topic 8: xxxx](#)

Attendees (Name, Affiliation, Email)

1. Jeff Carver, University of Alabama, carver@cs.ua.edu
2. Daniel S. Katz, University of Illinois Urbana Champaign, d.katz@ieee.org
3. Samuel Grayson, University of Illinois Urbana Champaign and Sandia National Laboratories (grayson5@illinois.edu)
4. Gil Speyer, Arizona State University, speyer@asu.edu
5. Dhruvil Deepakbhai Shah, Arizona State University, dshah47@asu.edu
6. Rebecca Belshe, ASU, rbelshe@asu.edu
7. Chris Reidy, University of Arizona,, chrisreidy@arizona.edu
8. Jason Yalim, ASU, yalim+SE4RS@asu.edu
9. Fang Liu, GATCH, fang.liu@gatech.edu
10. Claire Kopenhafer, Michigan State University, kopenhaf@msu.edu
11. Robert Caddy, University of Pittsburgh, r.caddy@pitt.edu
12. Ian Cosden, Princeton University, icosden@princeton.edu
13. Michael Gutteridge, Fred Hutchinson Cancer Ctr, mrg@fredhutch.org
14. Jacalyn Huband, University of Virginia, jmh5ad@virginia.edu
15. Weronika Filingier, EPCC, The University of Edinburgh, w.filingier@epcc.ed.ac.uk

16. Kevin Menear, National Renewable Energy Laboratory, kevin.menear@nrel.gov
17. Sandeep Puthanveetil Satheesan, University of Illinois Urbana-Champaign, sandeeps@illinois.edu
18. Lauren Milechin, MIT, lauren.milechin@mit.edu

Paper Session

"The Changing Role of RSEs over the Lifetime of [Parsl](#)" by Daniel S. Katz, Ben Clifford, Yadu Babuji, Kevin Hunter Kesling, Anna Woodard and Kyle Chard ([paper](#), [slides](#))

- [Parsl](#) is a library for parallelizing Python code and writing workflows
- A key feature is that Parsl can exploit parallelism in programs & workflows
- Healthy, growing project, but that can strain developer resources
- initial phase: lots of development, little maintenance. End phase: little development, lots of maintenance
- Want to document lessons on how to make sustainable communities of software
 - Going from 1 to 2 developers requires processes and documentation
 - Started using developer meetings to make big changes to the codebase
- Project stages:
 - 1. Concept testing (initial development)
 - 2. Testing with initial users (continued development)
 - 3. Expanded usage, support (some development)
 - 4. Community usage, (sustained maintenance/effort)
- Types of programmers:
 - Research programmer (initial prototyper)
 - Software Developer (focus on maintainability)
 - User/developer (not dedicated to project, but might contribute)
 - Collaborating developers
 - Defining interfaces is a key challenge
- Lessons/future work
 - Don't be afraid to revise previous decisions
 - Going from 1 to 2 developers is a good time to revise previous decisions
 - Research software sustainability is hard, no simple answers
 - Do other projects (and other kinds of projects) have this same experience?
 - Boundaries between types can be fuzzy
- Did you ever have to throw away and start from scratch? Kind of. We called it a different project

"Wanted: standards for automatic reproducibility of computational experiments" by Samuel Grayson, Reed Milewicz, Joshua Teves, Daniel S. Katz and Darko Marinov ([paper](#), [slides](#))

Repo: <https://github.com/charmoniumQ/execution-description>

Large Group Discussion

Questions for Discussion

1. How to handle platform change (hw and sw)?
 - a. Your users will notice them for you and maybe also be able to fix
2. Most relevant part to learn from other projects?
 - a. Governance
3. Other possible projects to learn from?
 - a. Linux (too big)
 - b. Apache
 - c. Exascale (doesn't know either, mostly just continuous funding)
 - d.

Discussion/Speed Blogging Topics

Topic 1: How to get researchers to use good practices?

Proposer: Robert Caddy

People Interested

- Samuel Grayson

Stuff we talked about:

- Some institution has weekly meetings to discuss best practices, can invite other people, can argue about a specific best practice
- Talking to people 1:1 is more reliable than addressing a group, but hard to scale
- At one institution, everyone has to take “practical HPC” before they get an allocation. But it's so much information people forget when it comes time to actually use it.
- Every two-weeks, some institution has hack sessions dedicated to code quality, which makes small-ish improvements on the code
- Every week, some institution has a meeting to look at issues, pull requests, and deal with them going forward
- Could invite/consult engineers outside the research group to help improve code quality
 - Get self-assessment and next steps <https://bssw-psip.github.io/>
 - Identify the smallest, incremental next step
 - Scales up well, because self-evaluation
- Git and testing are possibly the most essential
- Can ChatGPT or linters could be helpful
- Funding agency/journal might require certain software engineering practices

- Cookiecutter templates with best practices and argument parsing can be helpful. But can be a hindrance if they are inadequate. Need to document(!) templates.
- Need library information to archive software source code

Target and reach out to new initiatives, offer help to set up best practices, helpdesk tickets

Elevator pitch (could be helpdesk ticket):

- Identify what are their goals? Audience? Make suggestions based on that.
- How long do you want this code to be in use?
- What were your issues/challenges? Has this ever happened to you? Suggest a practice based on that.
- Gauge their level of interest and comfort in having the conversation.
- Need to have a reason that directly relates back to science or values scientists already have.
 - Reproducibility and correctness => Several retractions due to software bugs! Productivity => It could save the next grad student a lot of time if the current grad student does something like documentation.
 - Could also help using peer-reviewed publications, which scientists are already used to

Challenges:

- Choose your battles! One thing at a time. Walk before you run. Choose the “next best” practice (what’s worst).
- Hard to scale up training best practices
- Hard to justify refactoring legacy code
- Hard to find software engineering resources that are not just learning how to program. Could be book books (not textbooks): Pragmatic Programmer, Clean Code, Working with Legacy Code
 - Do we need to have books or training material from our own community?
- Working with group instead of individuals
- Hard to work with distributed groups

Possible solution

- Weekly hackathon dedicated to improving code quality. They see the benefits of better practices in person and have time to focus on them.

Topic 2: Creating/sustaining RSE teams

Proposer: Claire Kopenhafer

People Interested

- Sandeep

Notes:

<https://docs.google.com/document/d/1CcgqaTnLx309YjLNjUXd9wOuMvWmd5IUkDZ8p7hEtBQ/edit>

Topic 3: xxxx

Proposer: xxxx

People Interested



Topic 4: xxxx

Proposer: xxxx

People Interested



Topic 5: xxxx

Proposer: xxxx

People Interested



Topic 6: xxxx

Proposer: xxxx

People Interested



Topic 7: xxxx

Proposer: xxxx

People Interested



Topic 8: xxxx

Proposer: xxxx

People Interested

