Collegeville2021: Day 1 Breakout Notes, Room 3
July 20, 2021

Document source:
https://docs.google.com/document/d/1A2b50RtHSAclvvxJBf11fdsLSKZtNLAGpabV-tYH_TQ/edit

Overview: The content your discussion group creates in this document will be synthesized in a blog posting for https://bssw.io

Instructions:

1. Pick one person in your discussion group to create a new copy of this Google Doc
2. Make a copy of this template in a new Google Doc (the person from step 1)
3. Share the edit link to the document in step 2 with others (copy and paste into Zoom chat)
4. Co-edit the document: Can have one lead writer with others modifying, or another approach
5. Send the document to Mike Heroux at the end of the session by email (mheroux@csbsju.edu)

Add group member names for anyone who wants attribution in the blog post:

1. Name, affiliation as you would like it listed in the blog post
    - **Jim Pivarski**, Princeton and IRIS-HEP (@jpivarski, pivarski@princeton.edu)
    - **Gerasimos Chourdakis**, Technical University of Munich (@MakisH, chourdak@in.tum.de)
    - **Jay Lofstead**, Sandia (@gflofst, gflofst@sandia.gov)
    - **Cody Balos**, Lawrence Livermore National Laboratory (balos1@llnl.gov, @balos1 on GitHub)
    - **Lois Curfman McInnes**, Argonne National Laboratory (Github: @curfman)

Add a paragraph describing each person's software team background and experience. Emphasize experiences that inform your opinions about challenges, and how to improve software teams. Can add paragraphs in parallel :)

- Jim: library developer for end-user data analysts in a crowded field of small projects + a few big ones. Mostly Python analysis tools that end-users combine into an analysis. Some of these projects are funded under IRIS-HEP (multi-institution NSF project), more are corralled under Scikit-HEP (brand/umbrella organization). Some of these projects, such as the one I work on, have as many as 5 core developers at a time, though only 1–2 are funded as RSEs, others are IRIS-HEP fellows and GSoC students (primarily undergraduates), and there's a long tail of "pro bono" contributors, from "core" to "drive by pull request." We need to be very open to changing teams and need to homogenize mismatched assumptions. Since we develop libraries, we don't know the application context and can't fully optimize performance for specialized cases, so most of the work is streamlining the interfaces to make it for the end-users (to simply use or to optimize).

These libraries do a variety of analysis tasks: I/O, data structure manipulation, histogramming, fitting, interfaces to machine learning, etc. I don't know of anyone in this ecosystem (my project or related ones) who follow agile formally, but we do have shared assumptions about how to evolve code with issues → pull requests, maintaining a clean branch, continuous testing and linting, and automated deployment. We follow many "best practices," but not a strict methodology. As BDFL on my two main projects (Uproot and Awkward Array), I meet with each regular developer on Zoom once a week, but there's a long tail of casual code-committers that I interact with only on GitHub, Slack, Mattermost, and Gitter.

- Cody Balos: I am a library developer for a math library (time integration) with a small (4 people) development team (I am the only one with a formal computer science background, the other members have a math background). Our development is milestone/deliverable driven, we don't formally follow agile or any other methodology. However, our team does have excellent communication which I think (coupled with our small size) allows us to be successful despite following a formal development process. We communicate in regular meetings, adhoc meetings, slack, email etc. All decisions on API design are made as a group, and implementation details are discussed in pull request reviews (conducted by at least one other team member but usually 2). The primary points of disagreement are usually on small things such as code style (we do not have a code formatter because no one has had the time to set it up). We do have automated CI. Our team size does mean we have to carefully prioritize work items to meet milestones and deliverables, which I think has led to some technical debt buildup. There can be competition between what we must do, and what we would like to do (or consider optimal) in order to work for a diverse set of applications and meet the deliverables for many different sponsors. Also, since our library has a long history, it has many users who we support but are not sponsored to support directly.

- Gerasimos Chourdakis: preCICE coupling library.
    - team of students + PhDs + postdoc (Munich and Stuttgart), software project is central in the development, but people also have other tasks, development happens in a semi-agile way: regular meetings, releases, etc, but development takes higher or lower priority depending on the time in the semester.
    - Funding mostly from projects that are not specific to development.
    - Communication: Gitter chatroom + weekly 30min telco (on BigBlueButton) + coding evening with students in Munich every month (pre-corona) + coding days every three months. Regular feature releases every six months.
    - We have to work with and support multiple different frameworks (OpenFOAM, FEniCS, deal.II, CalculiX, …).
    - No clear connection between funding sources and who works on what: many work packages require collaboration among different people.
    - Also a lot of unofficial work to support users, maintain documentation, write tests, …

    - We are not users of our software, we want others to use it so that we can get feedback on the methods and improve the implementation.
    - Trying to involve the community and make them contribute. Difficult as most users don't come from a strong programming background. Compartmentalization of software (core library + bindings + adapters + tutorials) helps.
    - Have learned a lot from xSDK and Linux Foundation Core Infrastructure Initiative
    - Team starts getting bigger. More exciting, but also difficult to understand each other without the social interaction right now.

- Lois Curfman McInnes: currently working to coordinate collaboration by a variety of teams who are developing software technologies in the DOE Exascale Computing Project; prior experience in leading the xSDK project (community collaboration to advance the combined use of math libraries developed by different groups) and developing open-source software for large-scale problems based on partial differential equations and related optimization problems (PETSc/TAO libraries); experience in collaborating with applications scientists who are users of library capabilities (and who motivate the development of new functionality). Various projects include collaborators at different institutions, in different countries, with different backgrounds and levels of experience.

- Jay Lofstead: currently working on too many projects focusing primarily on data management (IO libraries, analytics support, workflow infrastructure, reproducibility supporting software infrastructure, and tons of related things). I spent 10 years in industry as a production software engineer including my last job running an Internet startup within a corporation that transformed the software business. Having lost control of the project, I went to grad school and moved into the science realm with no regrets. I think the biggest problems we have are around skills. Too many application scientists have basic programming skills and believe they can do anything. Instead, they do not have the discipline and training to follow proven approaches that make software workable in the long run and do not believe they need to ask for help from anyone. This applies to all aspects of the software. By including professional software engineers, we can start to address this--if they are treated as peers rather than subordinates. THe other issue I deal with often is how to incorporate interns and other new staff into projects effectively. This is tricky to get value for the time investment. For new staff, spending a couple of months to get something to compile might be ok, but for an intern on a 12 week program, it is disastrous. These kinds of configuration management tools and techniques should be common. Additional issues relate to automated regression testing and integration testing. This not a side job, but is often treated as one. It needs to be made a priority and it will vastly improve quality--assuming they truly get veto power over new things because they break other things in spite of the prominence of the submitter.

Discuss as a group the most important challenges you see facing scientific software teams. Try to refrain from solving the challenges (that's tomorrow and the next day). Summarize discussion in outline form.

- Point …
    - Sub-point …

Initial notes from Jay:

- Different perspectives: library vs application
    - Difficult to distinguish between performance implications of library from specific application
- When to adopt best practices as a research exploration becomes a useful tool?
- How to incorporate new people into a project at different maturities
- What sort of portability/specific compiler/language (GPU/ASIC/FPGA) features should core rather than abstracted?
- How to isolate portions of the code when conflicting library versions are needed for different portions?
- How do we offer better support for reproducibility and traceability for results?
- When should code be made public?
- When should data sets be made public?
- Do different open source licenses matter to scientific software users? If so, what should be used and when?
- How do we encourage issue tracking as a communication tool rather than other techniques?
- How do we avoid a project fork when disagreements happen? (This is the worst part of open source software to me (Jay)--so many slightly different versions of things because forks driven by disagreement about what should be in the feature set)
- How do we fund long term viability of software and data?
- How do we get funding to incorporate professional-level software engineers onto scientific software teams to aid sustainability, quality, and development speed?

Challenges:

- sometimes not a direct connection between what needs to be done (new functionality, support for users) and commitments to funding sources
- Comparison with industry: we don't operate as a large software company (specific developers working on specific tasks) but as a startup company (pool of funding, everybody does their best to develop whatever they can)

About 20 prior to the end of the session, around 1:40 pm CDT, try to reach consensus on 3 - 5 high-level challenges your team identified

- **Challenge 1:** disconnect between what is highlighted in funding proposals about what the funding is for and what is actually needed to make a software product successful. Also in what's considered during hiring.
    - Testing, deployment, PRs & merging commits, user support, etc
    - **Career paths:** for junior members to justify work on what needs to be done when it won't be recognized as valuable for their careers
    - **Angle related to software teams:** Need team members to handle all of this very important software work, yet funding agencies do not necessarily provide opportunities to pursue funding for these activities

-

- **Challenge 2:** awareness of different considerations between libraries and applications: how to characterize performance (characteristic examples vs THE specific instance), the **interaction with a community**, which is more essential for library developers than applications. **APIs are joint-responsibilities**; applications have these issues on one end (dependencies), libraries on both (dependencies and dependents). For dependents, dealing with backward compatibility is an issue. Mocking in tests. Semantic versioning.
    - **Angle related to software teams:** Determining how to collaborate on the intermediate spaces betweens software providers and clients;
    - Wanting to integrate the team with the community
        - consider software ecosystem perspectives
    - APIs are collaboration! Sometimes, interfaces are mirrored in **liaisons** among teams.
- **Challenge 3:** Remote communication can be ambiguous: context in a code review, “thickness of communication channel” (no conversations over lunch, etc.),
- **Challenge 4:** Size of teams, meetings: diminishing returns. Subgroups, hierarchy, sub-teams.
- **Challenge 5:** Scientific teams: backgrounds are more diverse (CS, domain scientists, mathematicians); speaking in different languages. Attempted solutions involving teach-ins are hard to justify when the bottom line deadline looms.