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:

Name, affiliation as you would like it listed in the blog post

1. Reed Milewicz, Sandia National Laboratories, rmilewi@sandia.gov
2. Jacob Moxley - Sandia National Laboratories - jcmoxle@sandia.gov
3. Ben Cowan (benc303) - Formerly with Tech-X Corporation, moving to Pilot AI
4. Sarah Osborn (osborn9) - Lawrence Livermore National Laboratory, osborn9@llnl.gov
5. Robert Jacob, Argonne National Laboratory, jacob@anl.gov

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 :)

- Reed Milewicz -
- Jacob Moxley - I am on Reed Milewicz's team researching software team dynamics and work on developing CI/CD pipeline tools for scientific software teams at Sandia. The biggest difference between computer scientists and other scientists/engineers is the way they view software. Often non-computer science people view software as a means to an end which can contribute to a culture of "Get it working now so we can publish." This can come into tension with computer scientists' desires to use software best practices.
- Ben Cowan - Worked on large-scale, high-performance, closed source, physics simulation codes
- Sarah Osborn - Currently am part of two different types of software teams, one is focused on mathematical software (an individual library, hypre, with a smaller team and the xSDK, a large team spread across many institutions). The other type of software team is devoted to resiliency of critical infrastructure – a small team developing a closed source software library. The team here is a collection of CS folks, mathematicians, and subject matter experts (e.g., power engineers) who often view software as something that is purchased from a vendor (with a nice, pretty GUI).

- Robert Jacob - Worked on global climate models for almost entire career. Currently leading the infrastructure group in DOE E3SM project. Have worked in software teams of various sizes but never larger than 10 on a specific piece of software.

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.

- Bridging the divide between the scientific software community and the world of conventional software development, which has many good ideas which we could translate to the scientific domain
- Offering competitive salaries to acquire and retain software engineering talent
- Making software development plans that have any resemblance to what actually happens.
- Finding qualified staff
- Keeping everyone on the team involved and interested in the work.
- Planning and coordination, making a software development plan and sticking to it.
- I guess I would also say finding time for everyone to work together as a team. It seems that no two people have time to focus on a software project together at once, because a lot of the software developers are scientists who have many non-software things to do. When things get done, it's that one person goes off and does it alone and then hands off to the next person when they have to switch focus. So there's very little team programming. At most you can find two people to work together, maybe three. (E3SM is also a distributed team with people over 7 labs so that also makes true team programming hard).
- Resolving the tension between code development and publishing, the clash of incentives.
    - Robert: Our project is a mix between scientist-programmers and (non-scientist)-programmers, and the scientists feel they don't get adequate credit for their software work. We have managed to bring onto the project -- and the funders are fine with this -- people who just write software rather than publishing papers.
    - Jacob: Could software citations help with this? I've started to see this in some of the conferences, where citing software is becoming more common.
    - Robert: What we really want is individual credit. Like if your name is on a paper, and you get cited, that affects all kinds of indices and metrics used in promotion.
    - Reed: Jacob and I are involved in data mining studies of scientific software projects, and identifying who is involved in a project and what they're contributing is very challenging. Some people are visible from things like publications, but there are often many more people involved in the software development to various degrees, and data on all those folks is rarely readily available
    - Robert: And funding agencies also have to count software contributions and not just publications as well. That has to factor into their decision-making and into how their success is judged..

    - Sarah: Same with managers and project sponsors looking to reach milestones. Researchers want to contribute to software, but they also need to answer questions and make progress towards those goals.
    - Ben: Think about a piece of software like Hypre. Say you use hypre, an underlying library, to do some amazing work with your software. Does hypre get sufficient credit for that?
    - Sarah: This comes up for us on the Hypre team. Teams have needs and feature requests but we may not have a pot of funding to meet those requirements.
    - Jacob: Part of this is coming up with rigorous metrics to justify a need for funding, something that allows funders to hold teams to account by checking whether they actually meet those metrics.
    - Reed: Establishing that your code is actually being used and demonstrating engagement metrics is very challenging, that's something a lot of the teams we work with have been asking for lately.
- Need for community engagement and outreach in the development of community scientific software
- (Sarah) Onboarding new staff members, even just interns. We can spend 12 weeks teaching them best practices in C++ and things like that. How do we teach best practices but also giving them a quick start into jumping in and contributing to the code.
    - Robert: DOE has all sorts of programs for undergrad interns (12 week programs), but I rarely take advantage of those because of the amount of time I'll end up spending just spinning them up.
    - Ben: And everything is changing. C++ is evolving much faster these days! Then you have CUDA! AMD vs. Intel in the battle for performance! There's a constant need to learn new things.
    - Reed: We find that teams struggle to keep up with the latest and greatest, you really have to stay in the loop on all these different changes in the computing landscape.
    - Ben: I find that you end up with one person who has the knowledge and expertise in an organization. The question is how to get that knowledge to the right people elsewhere.
- Work-life balance, cognitive overload, people are divided between too many different things
- In large organizations, researchers have many divided commitments to too many different projects. So many different distractions, a much greater management burden
    - Sarah: Siloing is a big problem in large organizations, lots of redundant efforts and missed opportunities.
    - Jacob: And at Sandia, for instance, I haven't met many people who could be described as full stack developers, they're all focused on their particular niche.
- Keeping the big picture in mind. Our team has lots of different individuals working on their own individual components, it's hard to maintain a line of sight to the big picture of the whole project.

        - Maintaining situational awareness is a big challenge. Scientific software today is more multi- across the board: multi-physics, multi-scale, multi-disciplinary, multi-institutional. It's hard to see the forest for the trees sometimes.
-

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**: Aligning incentives and rewards around code development and publishing. People often don't get adequate credit and recognition for their contributions to software relative to other forms of contribution (e.g. to publications).
- **Challenge 2**: Maintaining and growing software teams. Finding people with the right skills within the salary structures of institutions that develop scientific software. In this context, having better strategies for acquiring and retaining the right talent is also a growing problem. We're only just now beginning to address the needs of research software engineers (RSEs).
- **Challenge 3**: Scientific software development is increasingly large-scale and complex with development spread across multiple institutions. This necessitates better ways of communicating, collaborating, and coordinating -- especially across disciplines and institutions.