Day 2 Discussion: Collegeville 2021
Technical Approaches to Improved Software Teams

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, and GitHub ID (if available), as you would like it listed in the blog post

1. Jay Lofstead, Sandia, @gflofst
2. Johanna Cohoon, UT Austin, @jlcohoon
3. Robert Jacob, @rljacob, jacob@anl.gov
4. Sarah Knepper, Intel Corporation, @sknepper
5. Keith Beattie, LBNL, @ksbeattie

Discuss as a group the most promising technical approaches you see as opportunities for scientific software teams. Summarize discussion in outline form.

- Successful technical approach will reduce the amount of communication needed
    - Success would mean things should be more obvious, so you would look at less documentation/less documentation needed
- GitHub/GitLab/BitBucket - “software project management tools”
    - Communication forum (Issues, Discussion, comments)
    - The Pull Request mechanism
    - Collection of tools
    - Central location
    - Conversations about code are (visually) near the code.
    - Branching is easy
    - Has a Wiki, but requires GitHub account to edit (not everyone has)
- Kanban boards
    - E.g. in JIRA
    - Hopefully linked to github.
- Notion
    - Low code/no code software
    - Can take notes in it and can turn notes into database (lots of linking allowed)

    - Good for logging thoughts
- Confluence, or other Wikis
    - Something entire project team has access to. Not everyone is a developer and so may not have a github account for their wiki. Low bar to entry.
    - Documentation builds over time.
    - Open permissions so anyone (with access) can read/edit any page. Portions are world-readable for external users.
    - For documenting processes, procedures, plans, meetings, results, decisions. Not code.
- Keeping documentation updated
    - Regular review cycle: annual, monthly perhaps
    - Indicate when last modified
    - Define what kinds of changes need to be documented
- RSE or other dedicated personnel
    - Will (hopefully) be there for the long term
    - Reduces need for communication between several part time people
    - Focus on software quality
    - Be the expert on software dev and help educate the team
- Have specific goals in development
    - Feedback loop with users
    - Avoid unnecessary work by not doing things users don't need
    - Devote time to projects with clear payoff
- Onboarding process
    - How to introduce new people to a project
    - Scoping down (what parts you need to focus on, at least initially)
    - Consider any NDA or licensing issues they need to be told about
    - As first task, update any outdated section of the New Hire/Onboard guide
- Offboarding process
    - Often don't get that luxury
    - Make sure their knowledge is transferred, tasks re-assigned.
- Software written by non-software experts is passable, not great
    - Writing software - cooking analogy
    - Big educational component - understanding why/the value of processes
    - Documentation from the experts on how to start, standards to follow.
- Emphasize importance of the team
    - Best practices might seem like a detriment to individuals' productivity, but they benefit the team—show that team value
    - Need this to be reflected in evaluations (annual reviews shouldn't ding you for being prosocial)
- CI and code checkers
    - Give team members confidence to change code.
- Can make it work, but should you? Intersections with quality.
- Tools used
    - If can't pay for support, can't use it

        - Higher-level management (outside of team) decisions may impact
- Slack (or other chat services)
    - Incredibly critical for asynchronous updates (team spread over multiple timezones)
    - Virtual team can cohere by hanging out in a channel.
    - Good to have an “all questions welcome” channel.
    - Can’t always trust status indicators for people.
    - Can be difficult to set the culture - big request (1 hour effort) isn’t good to put in Slack
    - Can be helpful to turn off notifications to avoid distractions
- Office hours
    - Weekly Zoom room
- Point …
    - Sub-point …

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

- Approach 1: Shared communications infrastructure and a shared understanding of how to use that infrastructure (e.g. perhaps don’t use email AND Slack or email AND Github)
- Approach 2: Browser-based software project management tools (e.g., GitHub), including training and encouragement from management to use
- Approach 3: Project documentation (processors, procedures, meetings, code docs) that is clear, findable, modifiable by members of the project.
- Approach 4: Routine, automated CI and code quality checkers to give team members confidence in changing code.