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

July 21, 2021

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

- James Willenbring - jwillenbring - jmwille@sandia.gov - Sandia National Labs
- Jacob Moxley - jmox0351 - Sandia National Labs - jcmoxle@sandia.gov
- Weslley Pereira - @weslleyspereira - CU Denver - weslley.pereira@ucdenver.edu
- Gerasimos Chourdakis - @MakisH - Technical University of Munich - chourdak@in.tum.de
- Lois Curfman McInnes - @curfman - Argonne National Laboratory (curfman@anl.gov)
- Vadim Dyadechko - @vdyadechko - ExxonMobil

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

- CI/CD pipelines
    - Automation
        - Ex - reviewing things manually that could be automated
            - Static analyzers, formatters, linters, good compilers
        - Git, github have good automation tools. Surprised at under usage of these tools.
            - Many of these tools produce a flood of information. How can this be filtered?
            - Custom Git workflows
            - Custom “smart” comparators for floating point output
        - Automate otherwise manual reminders

        - Automating code writing - github co-pilot - start typing code, it starts to fill in complete blocks of code.
        - Automated testing, different HW/OS
            - Force tests to pass before merge
            - What about TDD?
                - Generally more difficult with research software if you don't know what to expect from the results beforehand
                - Hard to impose TDD on scientists not trained in SE and CS. May be missing prerequisites.
                    - Example: helps to have certain design skills
                - Instead of a separate effort for creating a regression test suite, you do 2 things at once.
                - Lower the bar and do something.
                - Appreciate having tests, but find it difficult to figure out exactly what to test (oracle).
                - How to apply Test Mocking to Scientific Software?
                - Large testing suites have drawbacks b/c when they fail there are so many places for it to go wrong. Hard to track down if it's hardware/different modules
                - Unit tests have a fair bit of overhead. A lower bar is regression tests. At least you have something.
                - Coverage testing - keep coverage the same or better
                - How to avoid re-testing features that are not targeted by a specific change? Maybe use CI/CD tools
- Team collaborative software - tools to enable collaboration
    - blackboards, etc
    - Effective communication software
    - Identity management (e.g. centralized RocketChat vs distributed Matrix, similarly for GitLab and other tools: which tools to use to collaborate across universities?)
    - Dependency management
    - Metrics
- Jira can provide some of the things that Barry mentioned in the panel this morning. Everyone has to be on the same platform.
- Need smart notification system that adapts to user preference.
    - Standardized notification format for email → let GitHub/GitLab/Jira/… send email notifications in a standard format and allow any compatible client to filter/prioritize without manual intervention
-

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: Team collaborative software - Big issues include buy in, too many tools to keep track of, getting everyone on the same page for problem -> assignment -> workflow -> solution - Examples: Jira, GitHub, GitLab
- Approach 2: CI/CD pipelines - Which are the easiest to automate?, How to task the correct people for different bug fixes, or feature requests? Which tools have the best support? Example tools: GitHub Actions, GitLab CI, Cron, customized tools
- Approach 3: Tools that assist developer in automated quality assurance - linter, static analyzer, code coverage, shellcheck, clang-tidy
- Approach 4: IDEs for development - don't get in the way, feel like using them - can have a personalized choice here, team tools need uniformity. Might be an area that would benefit from increased awareness of available features.
- Approach 5: Automating common workflows such as release through scripting a series of git commands. There is a "Git workflows" package. Also automating common team workflows/standards. For example, tools that ensure compliance with coding standards.