Software Engineering: An SC-Inspired Focus.

Chris Green

SCD Projects 2019-12-12

Thesis

It is currently very hard to make changes to established software—for new features or computing paradigms—and for experiments to move to new versions of physics tools with confidence. This could have been easier with systemic use of SE processes.

I want to start a discussion on what these processes might be, and how we could put them into operation.

What Are We Talking About?

- **Software Engineering (SE)**: “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.”[1]
  —Methodologies and procedures for the whole software lifecycle: requirements, design, development, testing and maintenance.
- **Verification**: Does the software do what it was designed to do?
- **Validation**: Does the software match user requirements?

[1] IEEE Standard Glossary of Software Engineering Terminology, IEEE std 610.12-1990, 1990.

Why are we talking about it now?

- Lots of SE-related events at SC:
  - Both reflecting and describing increasing (and increasingly systematic) adoption across the HPC community, including (inter-)national labs and academic institutions.
  - Collective experience with what works and what doesn't in similar scientific fields.
- We (SCD in particular and HEP in general) are part of the HPC commnunity, and competitors for resources in a demanding—and evolving—computing environment. If adopting (more) SE advances the scientific mission—and everyone else is getting on board—why *wouldn't* we want to look at what works and do it ourselves?

How Does SE Advance the Scientific Mission?

- Agility: analyses can be extended, algorithms improved, and scientific applications can meet new requirements with confidence—more results faster.
- Easier to demonstrate transparency, **reproducibility**[2] (*cf* **replicability**[3])—trend in HPC journals and conferences. Also a discriminator in HPC application development assistance and time requests.
- More confidence in results and conclusions through planning, programming best practice, and comprehensive verification processes.

[2] With the same data and code the results can be recreated.

[3] Scientific conclusions can be confirmed with new data and analysis.

Project and Line Managers: Considerations

- Diverse team composition: software engineers, applied mathematicians, specialists in algorithms, optimization, and problem domains.
- Group and ecosystem structure:
  - Based around Research Software Engineers[4] (RSE) vs project-oriented organization.
  - Combine project outreach and training with direct project contributions.
  - Serial *vs* (low-*n*) multiple attachment to projects, long term *vs* short term attachments. More than one entity noted that they have learned to avoid serial long-term attachments of RSEs to projects.
- Cultural and sociological issues.
- Advancement and career issues.

[4] Who is a Research Software Engineer: “RSEs combine an intricate understanding of research (most hold a PhD) with expertise in programming and software engineering.”

Project and Line Managers: Reported Useful Items

- Checklists for developer “life events”—joining a project or group, leaving it, *etc.*
- Policies for ongoing activities.
- Productivity and Sustainability Improvement Planning (PSIP)—SE process analysis, evolution, adoption.
- Acknowledgment & acceptance of early project overhead, and gradual or staged rollout $\rightarrow$ realistic timescales, more accurate planning.
- Provision for common needs—centralization for cost savings, knowledge pool, transferability *vs* fragmentation for appropriateness, flexibilty, *e.g.*
  - Build-and-test infrastructure.
  - Tool evaluation, budget, and purchase.
  - Policy, training, knowledge pooling, collaboration. . .

Project Leaders and Developers: SE Nuggets

- Agile methodologies great for incremental, staged rollouts: “Release early, release often.” When requirements evolve quickly, so must software—while remaining verifiable and maintainable.

  Scrum: most well-known Agile process (defined roles—scrum master, *etc.*, regular “sprints”). Can be too rigid (!) in arenas where people work on mulitple projects.

  Kanban (*lit.* “signboard”): more flexible, scalable—*e.g.* GitHub “project” boards. Multiple projects $\Longrightarrow$ multiple boards. Evolve with experience, improving requirements management (“Epic, Story”), task granularity and interconnectedness. Combine with *e.g.* **Test-Driven Development**.

- Increasing availability of workflow tools, free, freemium and paid.

Project Leaders and Developers: SE Nuggets

- Emphasis on achievability and usefulness of any process: evaluate, adapt, evolve.
- Useful to consider gross attributes of a software product: extensibility, portability, performance, maintainability, and verifiability. Understand interactions and tradeoffs.
- Understand and use—consistently—different levels of testing: unit, component, integration, regression, acceptance; verification *and* validation are both important.
- Plan and design for verifiability: **Test-Driven Development** (failing test $\rightarrow$ code that makes it pass), “meta-computations” (constructed problems with verifiable solutions), code coverage requirements.
- Usefulness of documentation at all levels: requirements and use cases, user stories, design, testing methodology and other process, user guide, code reference, and algorithmic content reference.

Conclusions

- We should have a coordinated and continuous approach to evaluating SE processes and tools, and driving the adoption of those found to work. Support via training, project planning and collaboration across the organization will be integral to standardizing throughout SCD's projects.
- Software projects we control should maximize their ability to adapt to new requirements—both from users and the computing environment—with confidence in the continuing correctness of the software.
- Applying improved SE practice and process to existing codebases is hard but possible.
- Developers' time and use of their skills are wasted through the recurring need to analyze, fix, refactor, and optimize overly complex and insufficiently tested code. I would rather spend that time finding ways to do things better and meet users' evolving needs with code that we can be confident will do the job.

Supplemental Material...

Supplemental Material

- SC Presentation Materials
  - Better Scientific Software.
  - Developing and Managing Research Software in Universities and National Labs.
  - Software Engineering and Reuse in Modeling, Simulation, and Data Analytics for Science and Engineering.
  - 2019 International Workshop on Software Engineering for HPC-Enabled Research.
  - Full proceedings.
- Case Studies:
  - The War Over Supercooled Water.
  - *Evolution of FLASH, a multi-physics scientific simulation code for high-performance computing*: DOI: 10.1177/1094342013505656.

Supplemental Material

- SE resources:
  - Better Scientific Software.
  - Interoperable Design of Extreme-scale Application Software (IDEAS).
  - The Xpert Network.
  - xSDK: Extreme-scale Scientific Software Development Kit—policies.
  - PSIP tools.
  - GitHub and Microsoft Project Integration and Automation.
  - Using Kanban Boards in Microsoft Project.
- Reproducibility:
  - SC19 Reproducibility Initiative.
  - Reproducibility for ACM-TOMS.
  - NISO Badging Scheme.