The curse of growing scales: from inception to successful community-driven software development

V. S. Mahadevan*, A. Siegel

Mathematics and Computational Science Division
Argonne National Laboratory, IL 60439
Email: mahadevan@anl.gov, siegel@mcs.anl.gov

Sustainable HPC software development requires careful planning, design and implementation from the ground up, as growing demands during the product evolution renders static processes that were successful at a smaller scale untenable, when scope and resource requirements expand. This white paper aims to address some relevant questions to tackle the curse of growing scales in science and resources (human and computing) during various stages of scientific software evolution.

INTRODUCTION

Sustainable HPC software development requires careful planning, design and implementation from the ground up, as growing demands during the product evolution renders static processes that were successful at a smaller scale untenable [1]. Even a powerful computational solver with an efficient implementation that is verifiably accurate, needs to be designed flexibly with good software engineering practices in order to be successful longer term and to build a community of dedicated users (e.g., FLASH, PETSc, Trilinos, FEniCS). Software processes such as SCM, SQA (unit/regression test suites), language and scientific library interoperability provide confidence, improve usability and encourage collaboration in the development of the tools. Additionally, extensive verification of code portability, especially on LCF machines and performance regressions are essential to continually maintain software robustness on both current and evolving future architectures.

The inception of compelling algorithms, approaches or methodology and its transformation into a complete product necessitates careful software design principles. In this short paper, we focus on relevant questions to tackle the curse of growing scales in physics resolution, funding needs, resources (human and computing), which are tied closely to each other [2] and hurdles that can impede success over long term (decades of man-years) development of software projects.

I Pilot scale

The initial stages of scientific software development in a project require rapid code changes and experimentation in design, motivated by several key factors.

1. Flexible interfaces that support feature extensibility,
2. Utilization of standard software design patterns when possible,
3. Support for multi-language interoperability (Fortran/C/C++/Python/Julia),
4. Test driven development (TDD) to considerably reduce delivery turnaround time,
5. Extensive unit and integration test coverage (e.g., gcov, lcov) and flexible test frameworks (e.g., Buildbot, Jenkins) to effectively catch regressions,
6. Static code analysis (e.g., clang, cppcheck, ROSE), eliminating memory leaks/corruption (e.g., valgrind) for a robust end product.
7. Application scope: Solving demonstration problems to identify numerical and performance model bottlenecks.

II Mini scale

The advancement of scientific software (complex applications or libraries) needs vastly different use-cases to continually improve and add feature sets that attract dedicated new users.

1. Verify configuration/build systems on different architectures for reliability,
2. Interoperate with verified scientific software libraries; increases dependencies while leveraging previous efforts to build a well-balanced scientific ecosystem.
3. Accommodating user requests and minimizing barriers for acceptance

*Corresponding author.
(a) Extensive documentation and examples for improved usability,
(b) Tutorials and user group meetings for closer interaction with developers,
(c) Utilizing social media (Google+, Twitter, Youtube) and dedicated websites to showcase success stories,
(d) Explicit online access to mailing lists, forums and developers for advanced one-to-one discussions that might blossom into collaborations

4. Application scope: Increased experimentation in terms of optimal computational algorithms and implementation of abstractions for general class of problems through continually evolving example sets and a range of complex applications.

III Macro scale
Keeping abreast with increasing resource requirements (funding, people and access to hardware), feature lists can grow in an open-source software environment in several positive directions, even unforeseen while drafting the vision at pilot scale.

1. Repository hosting on community-based websites like GitHub or BitBucket encourage feature submissions (PR) and code discussion/review processes,
2. Multi-disciplinary collaborative proposals with various use-case teams,
3. Publications and conference presentations to disseminate capabilities,
4. Hiring of talented postdoctoral candidates, computational scientists and software engineers to alleviate research, support and maintenance needs.
5. Application scope: Increased application resolutions in terms of advanced discretization methods, solver/preconditioner choices, and strong focus on scalability/time-to-solution for large-scale problems on LCF machines.

IV Distributed scale
This is the community driven distributed development workflow by leveraging collaborations lead to long-term, large-scale sustainable software development [3]. Several challenges do hinder progress at this scale.

1. Tackling the exponential inflation in libraries that interact, models that are coupled and hardware additions that complicate software portability; develop additional abstractions and re-focus extensibility to alleviate concerns.
2. Reward metrics: Currently, the merit-to-effort ratio is inversely proportional to software development growth, since increase in users necessitates more time, resources and funding. As a broader computational community, processes are necessary to gain credit for developing reproducible, scalable scientific software.
3. Application scope: Production ready, maintaining performance and software portability, coupled multi-physics/multi-scale applications with varying complexity to do predictive computational science.

SUMMARY
We have structured this whitepaper based on gradual scale advancements that are quintessential to most successful projects and have attempted to provide characteristic guidelines on achieving sustainability in scientific HPC software. It is imperative to note that there are no magic bullets [4] to address all the complex computational workflow needs of various scientific software development teams, and hence it is vital to identify key components that can be incorporated in these workflow processes to enable sustained increase in scientific research through better software productivity.

The Computational Science and Engineering Software Sustainability and Productivity Challenges (CSESSP1) workshop is an ideal avenue to discuss these hard issues and to provide best practices to the broader computational science community for achieving distributed scale, community driven scientific software development.

REFERENCES

1CSESSP: https://www.nitrd.gov/csessp/