# The Many Faces of the Productivity Challenge in Scientific Software

## Hal Finkel - Collegeville 2020

# Productivity Affects All Phases of Software Development...

# Concept → Design (Planning)

- Searching for information on how similar problems were solved in the past.
  - Recommendation: We need better ways to search both scientific literature and scientific source code.
- Assessment of the capabilities needed for the software.
  - Recommendation: We need more-comprehensive software metadata capturing not only build dependencies and licensing, but information on testing, development practices, anticipated funding stability, and planned hardware-support timelines.

# Design → Implementation (Authorship)

- Translating the planned design into an actual implementation accounts for a significant portion of the overall software-development effort.
  - Recommendation: We need more-intelligent programming tools to assist with the generation of code for scientific applications.
- One important aspect of the programmer-productivity challenge is compilation/iteration time.
  - Recommendation: We need improved software analysis and compilation turn-around time in order to increase developer productivity. In addition, we need better tools to help programmers of scientific software create and maintain tests.

# Implementation → Correct Implementation (Debugging)

- An implementation cannot be considered complete until it has been shown to be correct. This includes both verification and validation, and critically, both processes are difficult.
  - Recommendation: Debugging tools, especially high-performance, instrumentation-based tools, should be enhanced to work at large scale in HPC environments, and on large code bases.

# Implementation → High-Performance Implementation (Tuning)

- The transformation of a working piece of software to increase its runtime performance is often time consuming and requires specialized skills.
  - Recommendation: Better tools need to be developed to help developers model ideal performance and scaling on platforms of interest such that these models can be compared to observed performance. Compilers need to be improved to generate better code even in the face of complex abstraction layers.

# Isolated Implementation → Implementation As Part of a Larger Project (Integration)

- Integration of scientific software into a larger workflow can be challenging for several reasons. Often, the integration method has a significant affect on performance.
  - Recommendation: Work needs to continue on tools that generate high-performance interfaces between components written in a variety of relevant programming languages.
- Integration of scientific software can also provide difficult when the different components lack an ability to coordinate their use of shared resources.
  - Recommendation: Runtime systems, and operating systems, need enhancements to ensure that application components can communicate to collectively manage system resources.

# Implementation → Reproducible Results (Provenance)

- It is critical to the scientific process that the results of running scientific applications are reproducible (at least for some period of time).
  - Recommendation: Programming environments need to make it easy to collect, statically and dynamically, dependency, configuration, and version information for all applications.

# Implementation → Updated Implementation (Maintenance)

- A lot of developer time is sunk into maintenance tasks: Updating code to work with newer versions of programming environments and libraries, comply with updated coding guidelines, handle new usage scenarios, and so on.
  - Recommendation: Better tooling needs to be developed in order to handle language upgrades and perform API migration.

# Initial Implementation Developers → Replacement/Additional Implementation Developers (Training)

- An important consideration for all aspects of the productivity challenge for scientific applications is the reality that training is a challenge.
  - Recommendation: It is important that all capabilities created to address productivity challenges place an emphasis on being easy to learn (or, at least, it should be easy to get started).

# Implementation → The Next Implementation (Knowledge Transfer)

- No particular implementation will remain useful for ever. Eventually, new requirements will overtake the ability of the software to be adapted by mutation, and a new implementation must be created.
  - Recommendation: Systems must be developed, perhaps making use of ML technology, to make it easy to keep both detailed and high-level documentation in sync with the code and complete.

# Conclusions...

- When we think about programming productivity, we often think only about the process of writing code, after the design has been decided, ending once the code meets some definition of "working."
  - Don't do this: We need to consider the entire software life cycle!
- There are investments we can make now to improve productivity across the entire life cycle of scientific software.

# Acknowledgments


- ALCF, ANL, and DOE
- ALCF is supported by DOE/SC under contract DE-AC02-06CH11357

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.