

Observations about Software Development for High End Computing

Jeffrey C. Carver¹, Lorin M. Hochstein^{2,4*}, Richard P. Kendall³, Taiga Nakamura⁴,
Marvin V. Zelkowitz^{4,5}, Victor R. Basili^{4,5} and Douglass E. Post⁶

¹Department of Computer Science
and Engineering
Mississippi State University
carver@cse.msstate.edu

²Department of Computer Science
and Engineering
University of Nebraska, Lincoln
lorin@cse.unl.edu

³Information Sciences Institute
University of Southern California
rkendall@cybermesa.com

⁴Department of Computer Science
University of Maryland
{nakamura, mvz,
basili}@cs.umd.edu

⁵Fraunhofer Center for
Experimental Software Engineering
College Park, MD

⁶DoD High Performance
Computing Modernization Office
Arlington, VA
post@hpcmo.hpc.mil

Abstract

This paper describes observations about software development for high end computing that we have made from several environments. We conducted a series of case studies of different types of codes, from academic codes to codes from governmental agencies. Based on those studies, we have formed a series of observations, some common and some different across these environments. This paper discusses each of those observations along with the supporting information from the different environments.

1. Introduction

Computational scientists and engineers face many challenges when writing codes for high-end computing (HEC) systems. With the movement towards peta-scale machines and the increasing focus on improving development productivity, writing a good code is even more challenging. The DARPA High Productivity Computing Systems (HPCS)¹ project is developing new machine architectures, programming languages, and software tools to improve the productivity of scientists and engineers [1]. Although the existence of these new technologies is important for improving productivity, they will not achieve their stated goals if individual scientists and engineers are not able to effectively use them to solve their problems. A necessary first step in determining the usefulness of new architectures, languages and tools is to gain a better understanding of what the scientists and engineers do, how they do it, and what problems they face in the current HEC

* This work was done while the author was at the University of Maryland

¹ <http://www.highproductivity.org>

development environment. Because the HEC community is very diverse [6], it is necessary to sample different applications domains to be able to draw any meaningful conclusions about the commonalities and trends in software development in this community.

This paper discusses some of the similarities and differences found in ten projects taken from different domains as understood by researchers in the HPCS project. We have undertaken a series of case studies to gain deeper insight into the nature of software development for scientific and engineering software. These studies have focused on two different types of projects, (characterized in Table 1):

- ASC-Alliance projects, DOE-sponsored computational science centers based at University of Illinois Urbana-Champaign, California Institute of Technology, University of Utah, Stanford University, and University of Chicago. In the rest of this paper, the codes are referred to as *ASC Codes*.
- Codes from DARPA HPCS mission partners (organizations that have a vested interest in the outcome of and often financial sponsorship of the project), some of which are classified and would therefore be inaccessible to other researchers. Due to the sensitive nature of many of these projects, they must remain anonymous in this paper. In the rest of this paper, these codes are referred to as *MP Codes*.

Table 1 - Types of projects examined

	ASC Codes	MP Codes
# of projects	5	5
Environment	Academia (ASC-Alliance projects)	Mission Partners (DoD, DOE, NASA)
Classified	No	Some
Code size	200-600 KLOC	80-760 KLOC
Type	Coupled multi-physics applications	Single physics to coupled multi-physics and engineering

While these case studies are still ongoing, the results to date allow for some cross-project analysis to provide a deeper understanding of the state of the practice [2-5]. In this paper, we discuss some observations gained from this analysis.

2. Goals & methodologies

While the ultimate goal for the case studies of the ASC codes and the MP codes is the same (to improve the productivity of computational scientists and engineers), we have used a different approach for each type of code. The object of study in the ASC codes has been the individual programmer, (e.g. “Where does the computational scientist spend her time?”), while the object of study in the MP studies has been the project, (e.g. “Which factors determine project success?”). Table 2 shows a comparison of the goals of the two types of case studies.

Table 2 - Case study goals

ASC Codes	MP Codes
<ul style="list-style-type: none"> • Characterize which scientific programming activities are time-consuming and problematic • Characterize the common problems encountered by programmers. • Characterize the impact of technologies on developer effort. 	<ul style="list-style-type: none"> • Identify project success factors • Identify ways that successful projects manage risk • Identify productivity barriers that should be addressed by vendors • Develop a reference body of case studies

Table 3 provides an overview of the methodology used for each type of case study. In general, the approach for the MP codes was more comprehensive (longer questionnaire, on-site interviews, multiple subjects interviewed independently), and the approach for the ASC codes was more lightweight, which permitted quicker turnaround time when running the studies. Furthermore, each type of study collected different types of information. The focus in the ASC was lower level (i.e. more details about fewer things), while the focus in the MP codes was higher level (i.e. less details about more things).

Table 3 Methodology

	ASC Codes	MP Codes
Type	Ongoing	Retrospective
Interviewees	Technical leads	Projects leads, project staff
Overview	<ol style="list-style-type: none"> 1. Pre-interview questionnaire 2. Telephone interview 3. Generate summary document 4. Send summary document for approval/comments 	<ol style="list-style-type: none"> 1. Identify project and sponsors 2. Negotiate case study participation 3. Pre-interview questionnaire

	<ol style="list-style-type: none"> 5. Generate synthesis report across all projects 6. Send synthesis report to all centers for approval/ comments 	<ol style="list-style-type: none"> 4. On-site interview 5. Initial list of findings 6. Follow-up 7. Write report
Focus	<ul style="list-style-type: none"> • Product: attributes, machine target, history • Project organization: structure, staff, configuration management • Development activities: adding new features, testing, tuning, debugging, porting, effort distribution, bottlenecks, achieving performance • Programming models and productivity: choice of model, adoption of language, productivity measures 	<ul style="list-style-type: none"> • Goals, requirements, deliverables • Project characteristics, structure, organization and risks • Code Characteristics • Staffing • Workflow Management • V&V, Testing • Success Measures • Lessons Learned

3. Observations

Using the different methodologies and the different foci described in Section 2, we performed a cross-study analysis between the ASC codes and the MP codes to look for similar and different observations that can lead to deeper insight into the software development process. In this section, we present those observations along with the support (or lack of support) provided by each type of code (ASC and MP) during independent case studies. To facilitate the discussion of the observations, we have grouped them into high-level categories (each found in a subsection below) with a list of specific observations.

3.1 Goals and Drivers of Code Development

Code performance is not the driving force for developers or users; the science and portability are of primary concern: All of the projects studied were parallel programming projects; therefore code performance is clearly an important goal. But, the primary interest of the users and developers is the science, not fast, scalable code (except where fast, scalable code is required to meet the scientific objective). As long as the overall project performance goals were met, the science and portability are of greater concern than the last 10-20% of speedup. For example, a member of one ASC Code team stated

that metrics, like scalability, are recorded because the sponsor requests them, while “scientifically useful results per calendar time” would be a more appropriate productivity metric. Furthermore, the developers of the MP Codes indicated that because the codes are often used for decades, they focus more effort on portability than on speed and scalability for the current hardware platform. In fact, to the extent that an increase in performance can be achieved through new hardware, these developers believe that the portability of the software is far more important than the efficiency of the software.

Code success depends on customer satisfaction: For the MP Codes, success or failure depends on whether the code developers keep their customers (not always their sponsor) satisfied. Conversely, in the ASC Codes, the developers were their own customers, so they had no external customers to please.

3.2 Actions and Characteristics of Code Developers

Most developers are domain scientists or engineers, not computer scientists: The majority of the developers for the ASC Codes did not have any formal training in computer science or software engineering. On some (but not all) ASC Codes, the chief software architect had a background in computer science. The developers of the MP Codes found that it is easier to teach domain scientists and engineers how to write code than it is for computer scientists to comprehend the deep scientific or engineering phenomena being captured by the code – especially at the research level.

The distinction between developer and user is blurry: In the ASC Codes, there are no “external” customers whose needs must be met. The primary users of the code are the developers themselves, who were adding functionality to advance their own research. In some cases, there are external users of the code. But, because these codes may require additions or modifications to be useful, an external “user” may still need to do a certain degree of programming. The MP Codes have more external users than the ASC Codes, but the developers still constitute an important portion of the user-base.

There is high turnover in the development team: Because the ASC Codes are developed in academic environments, many project members (postdocs and grad students) are involved for only a few years (e.g., in one project, the two technical leads had been involved for less than four years). Most of the ASC Codes evolved from earlier

codes that were written by scientists who had long since left the organization. Conversely, most of the MP Codes had a core set of developers that remained with the project for the duration, often for a decade or more.

3.3 Software Engineering Process and Development Workflow

There is minimal but consistent use of software engineering practices: All ASC Codes exhibited the use of a subset of standard software engineering practices (i.e. the use of version control systems, regression tests, and software architecture). However, the developers of these codes did little defect tracking or documentation, beyond user guides. Conversely, the MP Codes did not show this type of consistency in the use of software engineering practices across teams. Each team made use of a few recognized software engineering practices, but there was no uniformity across projects.

Development is evolutionary at multiple levels: In both the ASC and MP Codes, the developers are working on “new science.” As a result, they do not always know the correct output in advance, providing numerous challenges for verifying the code relative to the phenomenon being simulated. The teams all use an iterative, agile approach, where new algorithms are implemented and then evaluated to determine whether they are of sufficient quality for current simulations. Even though the more rigid CMM approach may not be appropriate for many of the MP Codes, many projects claim to follow this approach, possibly as a result of pressure from sponsors.

Tuning for a specific system architecture is rarely done, if ever: None of the ASC Codes optimize/tune their code for particular platforms. For example, they assume that they are developing for a “flat MPI” system, and so do not optimize for machines with SMP nodes. One project member made a comment that was representative of all projects: “The amount of time it takes to tune to a particular architecture to get the last bit of juice, is considerably higher than the time it takes to develop a new algorithm that improves performance across all platform. Our goal is to develop algorithms that will last across many lifetimes. We are not really interested in getting that last little bit of performance.” The MP Codes did perform some tuning, but it was in conflict with their larger goal of portability (see Observation 1 in Section 3.1).

There is little reuse of MPI frameworks: Several of the ASC and MP Codes built their own frameworks to abstract away the low-level details of the MPI parallelism. However, these frameworks are built from scratch each time, rather than being reused from another system. One developer from an ASC Code explained this lack of reuse: “We have encountered other projects where people have said, ‘We’ll use class library X that will hide array operations and other things’, but all sorts of issues arose. These frameworks make assumptions about how the work will be done, and to go against the assumptions of the framework requires getting into really deep details, which defeats the purpose of using such a framework.”

Most development effort is focused in implementation rather than maintenance: In both the ASC and the MP Codes, most of the effort was expended during implementation, rather than maintenance. This distribution indicates that rather than being released and maintained like traditional IT projects, these projects are under constant development.

3.4 Programming Languages

Once selected, the primary language does not change: Languages adopted by the MP Codes do not change once they have been selected. The majority of the older codes were written in FORTRAN77. Some newer codes and many of the ASC Codes are in FORTRAN90, C and C++. C seems popular for handling I/O issues. Some of the ASC and MP Codes have also adopted Python to drive the application. However, on one project the Python-based driver was later abandoned because it increased debugging complexity and reduced the portability of the code.

Higher level languages (e.g., Matlab) are not widely adopted for the core of applications: The MP teams have not adopted these higher level languages for use in their codes. The ASC teams have restricted the use of these high level languages to prototyping numerical algorithms.

3.5 Verification and Validation

Verification and Validation are very difficult in this domain: While V&V is difficult in all software domains, it was seen as especially difficult for both the MP and ASC

Codes due to some unique characteristics. It is difficult for developers to judge the correctness of code because they often do not know the “right” answer. In addition, there are at least three places where defects can enter the code, making it difficult to ultimately identify the source of the problem: 1) the underlying science could be incorrect; 2) the translation of the domain model to an algorithm could be incorrect; and 3) the translation of that algorithm into code could be incorrect.

Visualization is the most common tool for validation: All of the ASC Codes provide visualization to allow the users to view the large amounts of outputs produced. For the experienced user, this visualization provides a sanity check that the code is behaving reasonably.

3.6 Use of Support Tools during Code Development

Overall, tool use is lower than in other software development domains: Both the ASC and the MP Codes tended to view integrated development environments (IDEs) as being too restrictive. Instead, they liked the flexibility of UNIX-style command-line tools, such as make and shell scripts, to use for building applications. A complicating factor to wide tool use is that the target machines do not support the development tools that are available to programmers in other domains. Conversely, developers from both ASC and MP Codes did make some use of HPC-specific tools such as performance monitors and parallel debuggers.

Third party (externally developed) software and tools are viewed as a major risk factor: Because the code developed by the MP Codes tend to take years to develop, they are not willing to put their development at risk by relying on a software package or tool that may not be supported in the future. Furthermore, the ASC teams were concerned with the portability of libraries, which decreased their likelihood of use.

4. Conclusions

In this paper, we have summarized our findings from a series of case studies conducted with ten ASC and MP Codes as a series of observation. Due to different environments in which each code is developed, some of the observations are consistent across code teams while others vary across code teams. Overall, we found high

consistency among the ASC Codes and the MP Codes. Due to the different environments and foci of these projects, this result is both surprising and positive. In addition, despite the fact that a large majority of the developers on these teams have little or no formal training in software engineering, they have been able to make use of some basic software engineering principles. Further education and motivation could increase the use of these principles and further increase the quality of scientific and engineering software that has already demonstrated its value.

Based on the positive results thus far, we have plans to conduct additional case studies to gather more data in support of or in contradiction to the observations presented in this paper. In future case studies, we will strive to investigate codes from additional domains, thereby allowing broader, more inclusive conclusions to be drawn.

Acknowledgements

This research was supported in part by Department of Energy contract DE-FG02-04ER25633 and Air Force grant FA8750-05-1-0100 to the University of Maryland.

References

- [1] Hochstein, L., Carver, J., Shull, F., Asgari, S., Basili, V., Hollingsworth, J.K., and Zelkowitz, M. "HPC Programmer Productivity: A Case Study of Novice HPC Programmers". In *Proceedings of SuperComputing*. 2005. p. 35
- [2] Kendall, R.P., Carver, J., Mark, A., Post, D., Squires, S., and Shaffer, D. *Case Study of the Hawk Code Project*. Technical Report, LA-UR-05-9011. Los Alamos National Laboratories: 2005.
- [3] Kendall, R.P., Mark, A., Post, D., Squires, S., and Halverson, C. *Case Study of the Condor Code Project*. Technical Report, LA-UR-05-9291. Los Alamos National Laboratories: 2005.
- [4] Kendall, R.P., Post, D., Squires, S., and Carver, J. *Case Study of the Eagle Code Project*. Technical Report, LA-UR-06-1092. Los Alamos National Laboratories: 2006.
- [5] Post, D., Kendall, R.P., and Whitney, E. "Case study of the Falcon Project". In *Proceedings of Second International Workshop on Software Engineering for High Performance Computing Systems Applications (Held at ICSE 2005)*. St. Louis, USA. 2005. p. 22-26
- [6] Post, D.E., Kendall, R.P., and Lucas, R.F., "The Opportunities, Challenges, and Risks of High-Performance Computing in Computational Science and Engineering", in *Advances in Computers*. p. 239-301.