Commit Quality in Five High Performance Computing Projects

Kapil Agrawal*, Sadika Amreen*, Audris Mockus*
*Department of Electrical Engineering and Computer Science
The University of Tennessee, Knoxville
Email: kagrawa1@vols.utk.edu, samreen@vols.utk.edu, audris@utk.edu

***Abstract*—High Performance Computing (HPC) has a long history of software development but relatively little is known about the approaches this community uses to create and maintain software. To close this gap we study the practices of using version control tools in five HPC production projects. We also contrast these practices to practices used in three distinct non-HPC open source projects. We first obtain version history of the projects from SVN, Mercurial, and Git. We then clean and process the data and use published material to construct three measures of code commit quality: the fraction of unique commit comments, their size, and the number of files per commit. Our results indicate relatively high but declining commit quality, and relatively large commits in HPC projects. We expect this work to highlight the differences among different software engineering domains and may lead to ideas suggesting good practices of using software tools in these domains.**

***Index Terms*—Version Control Practices, Software Development, Software Repository Mining**

I. Introduction

High-end computing faces several challenges: oversubscribed resources, tight budgets, and projects of increasing complexity. Many of HPC projects start as small one-shot research experiment, but, if successful, may grow rapidly and become extremely difficult to maintain.

In this paper we present our analysis on utilization of version control systems for software development of High Performance Computing projects. Our goal is to understand how HPC code is developed in terms of various metrics such as commit size, frequency of commits, commits per user etc. and, if possible, suggest better practices of commits along with tools that utilizes such data to help HPC software development.

To accomplish that we compare HPC and non-HPC projects by investigating various measures of the development practices derived from version control systems (VCS) of these projects. We decided to retrieve our data from large software development forges, such as, GitHub and BitBucket; however, we focus our analysis on projects that we do know to be from the HPC domain. While many HPC projects use these and other forges to host their version control and issue trackers, classifying projects on these forges as HPC or non-HPC is still a challenge based on our investigation. Furthermore, forges, such as, GitHub and BitBucket have huge numbers of repositories (commonly referred as repos). At present GitHub has in excess of 20 million and BitBucket in excess of 400 thousand repos[1]. Finally, we know colleagues from the HPC projects we selected who can help us interpret the findings as, for example, suggested in [1].

We conduct our analysis as an exploratory parallel case study [2], a method suitable to investigate novel phenomena. Our hypothesis is that the software development practices, in particular, the use of VCS, will differ between HPC and non-HPC projects. For our case study we, therefore, selected a small sample of HPC projects from the Innovative Computing Laboratory (ICL) at the University of Tennessee, Knoxville, and non-HPC log files from three projects hosted on BitBucket.

Based on various online documents on good VCS practices we first construct several measures of code commit quality. In particular, we consider the ratio of unique commits over total numbers of commit log messages, quantiles of commit message size, and quantiles of the number of files involved in a single commit.

We then present trends of the quality of commits for these projects over the time they were developed. We also investigate numbers of contributors and numbers of commits to correlate the commit quality with social, organizational, and project maturity aspects.

Our main contributions include:

1) **A case study comparing five HPC projects involving key production frameworks with three highly diverse non-HPC open source projects;**
2) **Introduction of code commit quality measures**

II. Related Work

Early HPC software is was, perhaps, the birthplace of software engineering. The considerations of software portability and productivity are clearly expressed in some of the the early work, see, e.g., [3]. More recently, unique aspects of software engineering for HPC were described in, for example, [4]. The work presented there is a summary of a fairly significant body of literature. It, however, focuses more on the one-shot research projects, and not the production projects we are primarily considering in this work. As most research projects tend to be smaller and shorter-lived than the production projects, the use of software engineering tools and practices may not be always as crucial. Perhaps because of this, empirical measurement based on the traces from such tools is not considered often in the HPC literature. We, therefore, focus on the production projects that have significant experience of using VCS tools and our findings should be interpreted only within that context.

Methods to use VCS data in empirical work have been embraced widely, see, e.g., [5], [1], [6]. For comparison of HPC and non-HPC projects we focus on the metrics measuring the quality of the most important attributes of the VCS: code commits. Better quality of commits both, indicates a more mature project, see e.g., [1], [6] and a more effective project, see, e.g., [7]. Quality of a variety of issue tracking and version control system attributes is considered in, for example, [8] . The practitioner community is eager to educate participants on what a good commit messages, e.g., [9] and properties of a good bug report are considered by [10]. One of the most important attributes of VCS is code commit message (or log). GitHub, for example, provides

[1]While a single project may encompass several repos or the same repo may be used in several projects, we use terms software project and repo interchangeably in this paper.

online warnings, if the commit log header exceeds 50 characters. There are numerous suggestions of how to craft the commit log message online. We use these recommendations to propose a measure of code commit message quality in Section IV-B.

III. Context

HPC software engineering is the practice of enhancing software to enable it to perform better on modern multi-core processors, GPUs, HPC clusters and supercomputers. We investigate analysis of some practices of Version Control System use in these developments. The advantage of version control software is that it keeps a track of the complete file history (why/who/what changed), allows for shared development, and supports complex workflows (multiple branches). The analysis on how well one has utilized version controlling on HPC has not been fully studied. Our goal is to find out measures and trends, of these developments made over a period of time and do a comparative study of HPC and popular non-HPC projects.

Our study includes a small sample of HPC projects from ICL at University of Tennessee Knoxville, and non-HPC log files from three projects hosted on BitBucket.

The projects included in the study are:

1) **OpenMPI [11]: The Open MPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners.**
2) **OpenSHMEM [12]: OpenSHMEM is a Partitioned Global Address Space (PGAS) library interface specification. OpenSHMEM aims to provide a standard Application Programming Interface (API) for SHMEM libraries to aid portability across multiple vendors including SGI, Cray, IBM, HP, Mellanox, and Intel.**
3) **PaRSEC [13]: PaRSEC is a generic framework for architecture aware scheduling and management of micro-tasks on distributed many-core heterogeneous architectures.**
4) **PLASMA [14]: The Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) project aims to address the critical and highly disruptive situation that is facing the Linear Algebra and High Performance Computing community due to the introduction of multi-core architectures.**
5) **MAGMA [15]:The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.**
6) **BitBucket Tutorial [16]: A documentation/training project to help user of BitBucket understand and effectively use various tools available (most forked).**
7) **Django-piston [17]: Piston is a relatively small Django (JavaScript framework) application that lets you create application programming interfaces (API) for your sites (most watched).**
8) **Linux kernel [18]: A project involving installation and documentation of how to work with Linux kernel.**

IV. Method

We conduct exploratory multiple case study, using both literal replication and a theoretical replication (see. e.g., [2]). The literal replication was done for five highly similar HPC projects that represent widely used frameworks for parallel computing. We also carefully select three very different non-HPC projects to help formulate hypotheses about what metrics (and why) may be different among them.

For theoretical replication we selected three extreme non-HPC projects from BitBucket: with most commits, most watchers, and most forks. Each project was significantly different in at least one important dimension. Linux kernel was very long-lived and very large project where we expected to see exemplary commit quality: it would be impossible to develop such a large and complicated project with huge numbers of participants without having rigorous commit practices.

BitBucket tutorial was, in contrast, not a software project but a documentation/training project to help users of BitBucket understand and use effectively various tools available at the forge. It included a practice exercise for forking, where users were instructed to fork the project in order to practice contributing to it via pull requests. As a result, it was the most heavily forked project in BitBucket. We expect it to have fairly low commit quality because many of the commits are done by new users training to use BitBucket.

The last project we picked was highly popular but recent project related to Django framework (in JavaScript). This was in contrast to a highly mature Linux kernel written in C and BitBucket tutorial. We expect the quality of commits to be much lower than for Linux kernel. The high popularity (in terms of watchers) can be partially explained by the extremely rapid evolution of JavaScript frameworks and watching the repository may be the only way to keep up-to-date with the rapidly evolving state of the project. We expect the quality of commits to fall in between the BitBucket tutorial and Linux kernel, because unlike the former it is a real software development project and unlike latter it is not very mature.

To operationalize commit quality metrics we study voluminous literature related to this topic available online. A search for "good commit logs" yielded more than 57 million results. We used the top five links [9], [19], [20], [21], [22]. The specific measures and the rationale behind them are described in more detail in Section IV-B.

The measures of project size (in terms of the number of commiters) are borrowed from earlier research, e.g., [5].

A. Retrieval of metadata

We started with a little study exploring the prevalence of HPC projects on BitBucket. We selected HPC repos from BitBucket using most popular keywords that were used by HPC community. Keywords were divided into three categories namely:

1) **Libraries: Includes various libraries such as numerical, Fast Fourier Transforms (fftw), Distributed memory Dense Linear Algebra etc.**
2) **Software Packages: HPC software packages like molecular dynamics (NAMD, CHARMM), Commercial Finite Element Package (ABAQUS, ANSYS) etc.**
3) **Generic Keywords: Keywords like quantum chemistry, Parallel Molecular Dynamics.**

We looked at the description tag and repositories names to match against these keywords. BitBucket has around 400,000 repositories, though our search for HPC repos ended up raking around 80 repos. As the results from the look-up did not suggest that we were able to have a representative sample of HPC projects, we focused on a case study of comparing five HPC projects involving key production projects with highly diverse non-HPC open source projects.

These projects (HPC/non-HPC) were implemented using different version control. For HPC we had to work with Git, Mercurial (hg) and SVN and for non-HPC we had Git and Mercurial.

B. Measures

The following statistics were measured from the data we obtained from the HPC projects.

1) **Total number of commits. This measure is used to obtain the context for the project, in particular, how much**

Repos	Authors	Time	Cmts / Unique Cmts	VCS
openMPI	116	2003-	20,540 / 20,016	GH
openSHMEM	20	2010-	1,010 / 999	GH
PARSEC	33	2009-	7830 / 7392	BB-hg
PLASMA	20	2008-	3,999 / 3,805	SVN
MAGMA	21	2013-	4,149 / 3,844	SVN

Table I
OVERVIEW OF HPC PROJECTS SELECTED FOR OUR ANALYSIS

Repos	Authors	Period	Cmnts / Unique Cmts	VCS
eniliolopez/linux	15K	2005-	446K / 442K	BB-git
tutorials.bitbucket	2.6K	2012-	6K / 5.5K	BB-hg
django-piston	33	2009-2012	254 / 252	BB-hg

Table II
OVERVIEW OF NON-HPC PROJECTS SELECTED FOR OUR ANALYSIS. THE FIRST PROJECT HAS THE MAXIMUM UNIQUE COMMENTS, THE SECOND PROJECT HAS BEEN MOST FORKED, THE THIRD MOST WATCHED.

effort [23] went into creating and maintaining the project. We also use it as a normalizing factor in commit quality measures.

2) **Number of authors in a project. This is an important characteristic of a project. For example, commercial projects tend have fewer contributors but more equal contributions among them, see, e.g., [5]. We observe it together with the commit quality to interpret the evolution of projects.**
3) **Number of unique commit messages. Based on good commit practices each commit message should be tailored to a specific commit. For example, generic commit messages such as "fix" or "fixed bug," or "initial commit" are strongly discouraged. For the mature projects we expect the number of unique commit messages to be very close to or equal to the number of commits.**
4) **The size of commit comments. Good commit practices suggest a specific format and detail related to commit messages. Having very small commit messages may indicate the lack of maturity, as they may not explain all aspects of the commit that are needed for others to effectively maintain the codebase. We expect more mature projects to have larger commit messages.**
5) **Number of delta - the total number of files modified or added in a single commit. Good commit practices strongly discourage completing several tasks in a single commit, even though it is often the most convenient approach for an individual developer to submit all their work at the same time. Such multiple-task commits are more likely to involve multiple files. We, therefore, expect commits with more delta to be associated with less mature projects.**
6) **The ratio of total number of unique commit comment strings over the total number of commits. A high ratio reflects a high quality of commit meaning that the commit comments were tailored to each commit. Lower ratio indicates that same comments were reused for new commits or were generic, which indicates that changes in the commit are not precisely and accurately documented.**

The size of commit comments, the number of delta, and the fraction of unique messages are all measures of the quality of the commit.

These factors help us predict, among other things, the overall quality of commits in repositories of HPC code. For instance, fewer delta per commit indicates that changes in code are committed frequently and reflects incremental build or modification of the project. More delta per commit suggests that the commit may have involved several tasks: not a recommended practice.

Tables I and II show an overview of the projects selected for our analysis.

V. RESULTS

A. Fraction of Unique Commit Comments

Figure 1 shows the first measure of commit comment quality defined by the ratio of number to unique commits (nUC) to the total number of commits (nTC) for each of the five projects under consideration in our study. The sudden hike in number of commits shows that, as expected, many files are committed at the start of the project and then the rate of commits gradually drops as the projects mature. A close look at the nTC and nUC lines which are the number of total commit comments and the number of unique commit comments show that they almost overlap yielding a very high nUC/nTC ratio within a range of 0.9 - 1.0. This indicates that the authors have made an effort to document the changes almost every time the code was committed to the version control system. However, Open-MPI had a fairly low fraction in the beginning of the project, which also happened ten years ago, suggesting that the practice may not have been as rigorously observed ten years ago. The plots show that the average life of the remaining project is about five years and the nUC/nTC ratios for all projects are above 0.9 for the entire time frame showing a sturdy sign of 'health' for the projects.

To compare the statistics we derived from the HPC projects, we selected 3 non-HPC projects from BitBucket having maximum user commits (Linux project), most forked (tutorials.BitBucket project) and finally most watched (Django-piston project). Figure 2 shows the commit comment quality for the stated projects. One of the obvious differences of the Linux project from the HPC projects is that there are no sharp peaks in the graph. nTC and nUC both rise gradually over time showing that this project has been incrementally built and documented. The nUC/nTC ratio (quality of commit comments) for non-HPC projects are not as consistent as the HPC projects. While the Linux project has a fairly high quality of commit comments the other two projects show deterioration over time.

B. Number of Delta and Comment Length

We describe commit comment size as the number of characters in each commit comment. The plots shown for the HPC and non-HPC projects show that the size of commit comments in the HPC projects vary between 200-1300 (figure 3) compared to 50-150 (figure 4) for the non-HPC projects. The average size of comments for HPC projects are therefore significantly larger than non-HPC projects. This reflects that more effort is put into documenting how the code is being built within the HPC community compared to the non-HPC community. However, good commit commenting practices require including the name of each changed file in the commit with an explanation of the nature of the change for groups of files. With more files involved in a change we expect their names (and explanations) to increase the length of the comment. As such, the increased size of the comment may be simply a result of more files per commit. Further investigation is needed to determine that.

We describe delta as the number of files changed in a single commit. The plots in figure 3 and 4 show us that HPC projects have a higher delta compared to non-HPC projects respectively. For example, delta has an average of 5-6 for HPC projects and is as high as 16 for one of the projects called PLASMA. The average number of files changed in non-HPC projects is approximately 2. This difference may be interpreted as a good practice by the non-HPC community where small incremental builds are committed or it maybe due to less dependencies between files, i.e. changes in

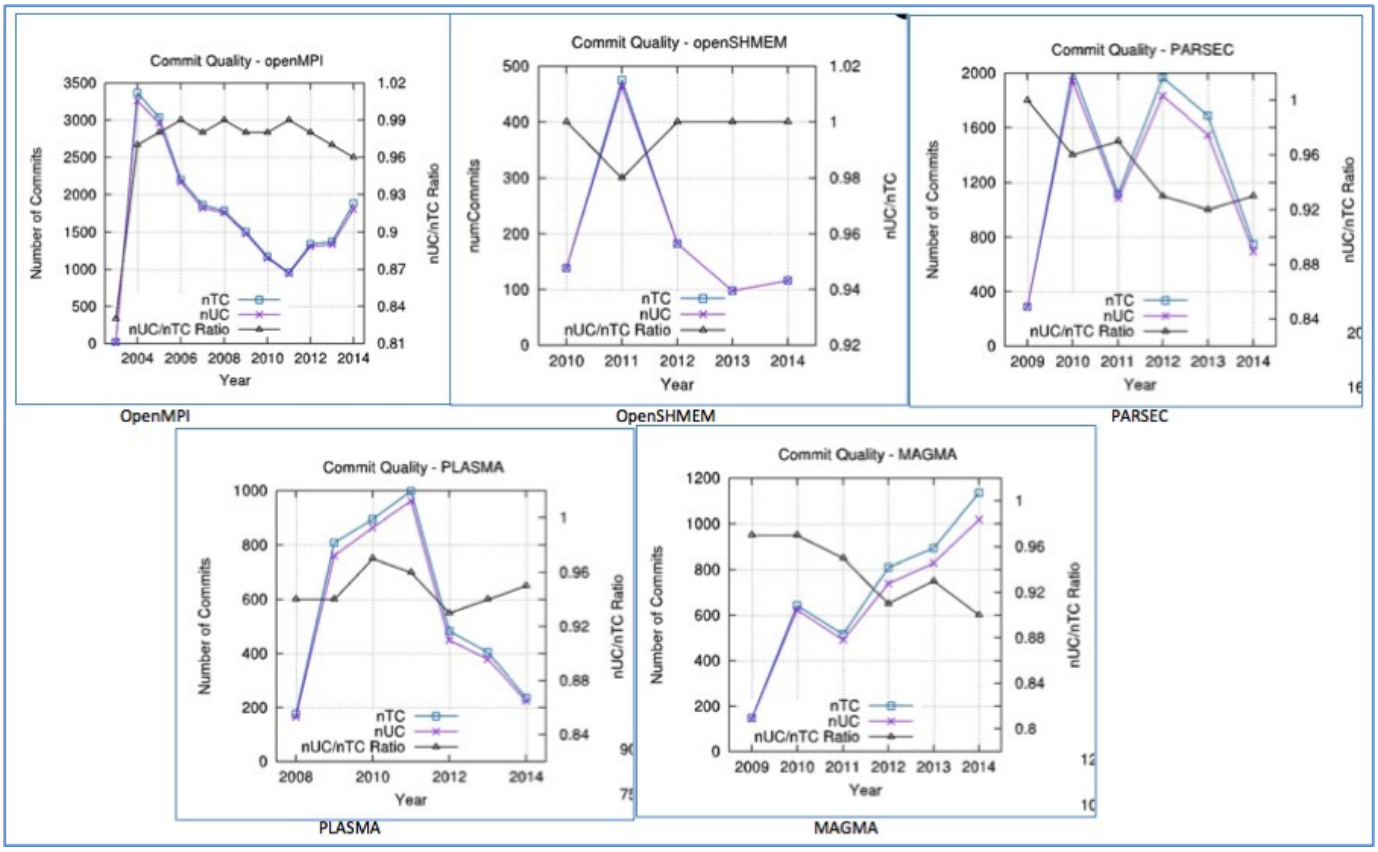

Figure 1. The trend in commit quality of HPC projects.

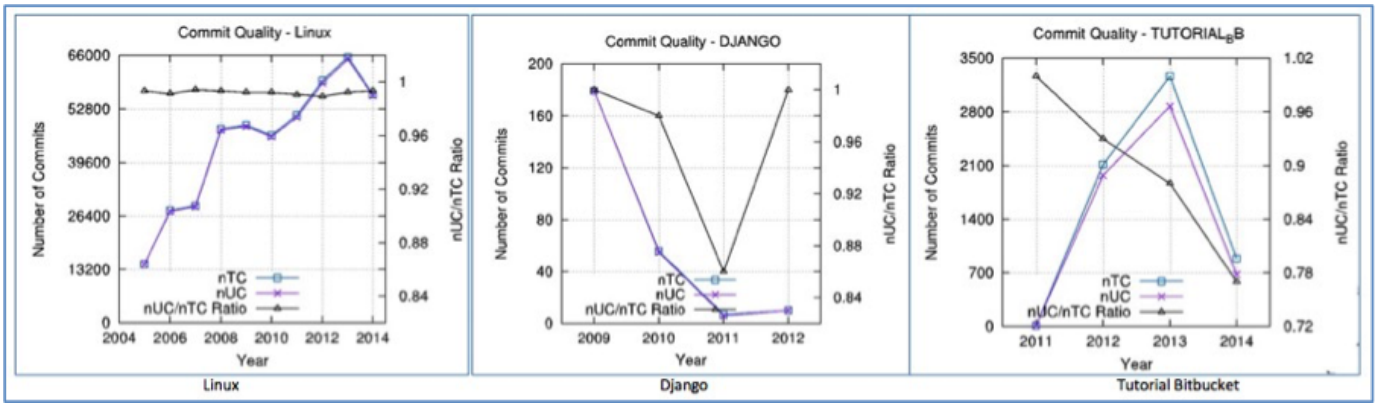

Figure 2. The trend in commit quality of non-HPC projects.

a certain file do not influence codes in other files. For example, delta for BitBucket tutorial is almost constant at 1 - this seems intuitively correct because we can expect not too many files to change since its inception. A higher delta can be interpreted in a negative sense because it might mean that large changes are committed in a single commit involve multiple tasks. This makes it difficult for developers to disentangle what changes to the code were done to, for example, fix a specific bug, from other activities that a developer may have been engaged at the time prior to commit.

C. Hypotheses

As described in Section IV, we expect the particular set of HPC projects to have commit quality higher than most open source projects, but, at the same time, lower than the very large and mature projects like Linux kernel. Based on the observations from our case study, in particular, on the fraction of unique commit messages and their size, we propose

***Hypothesis 5.1:* HPC production projects will have commit quality in terms of fraction of unique commit messages and message size to be above that of most open source projects.**

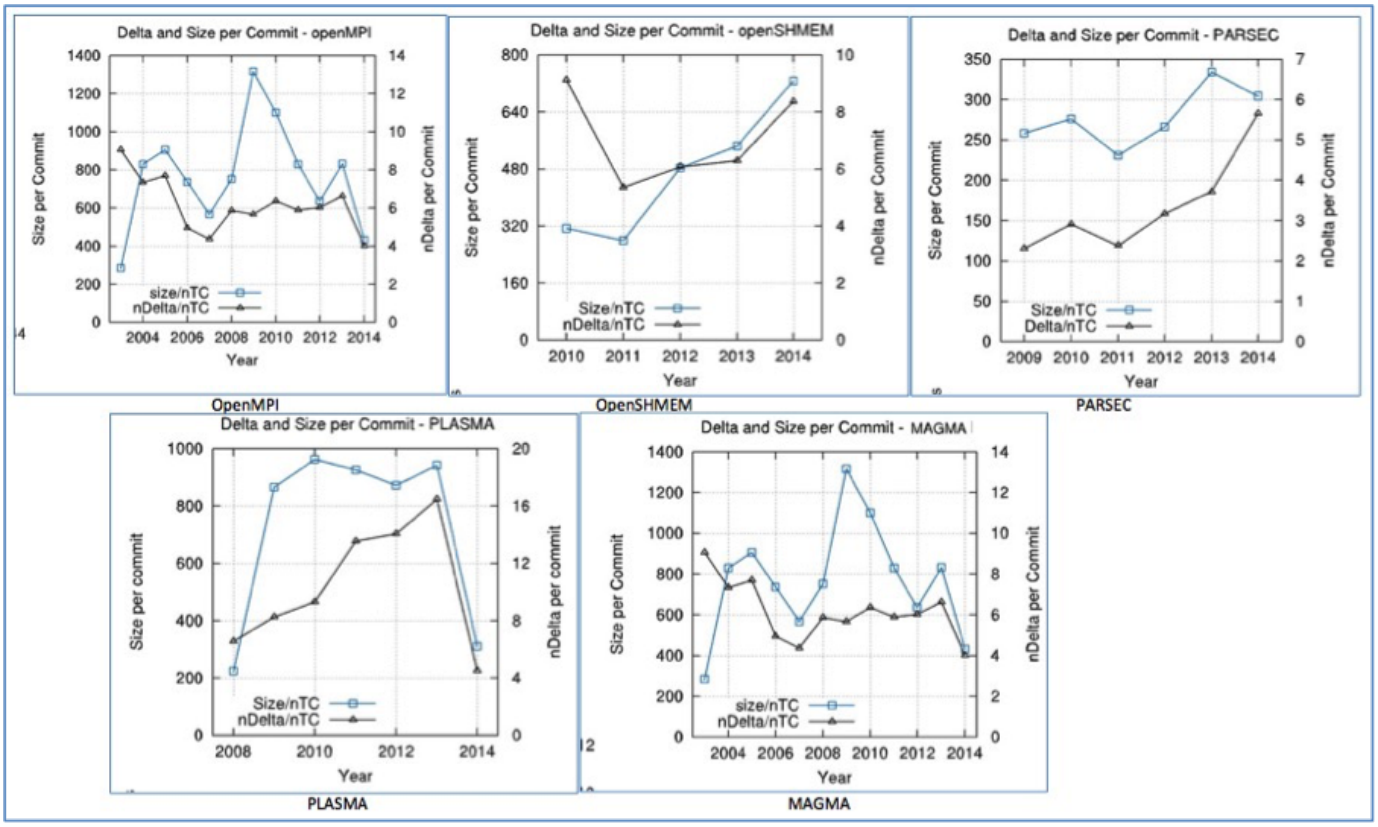

Figure 3. Size of commit comments projects for HPC projects.

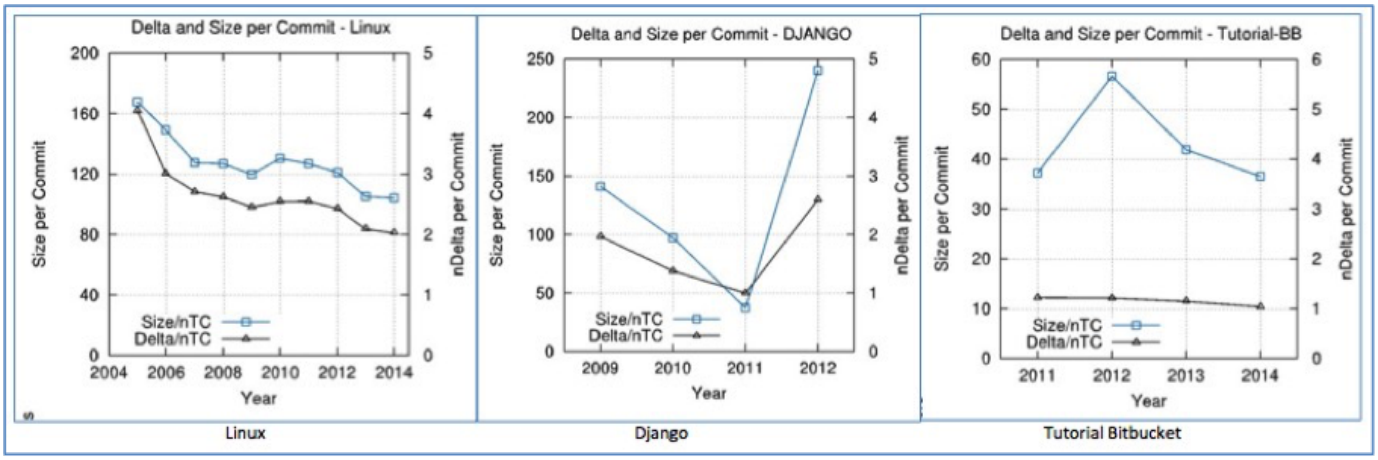

Figure 4. Size of commit comments projects for non-HPC projects.

We expect that HPC projects, by virtue of being funded from government grants or corporate funds, are likely to be more like commercial projects [5].

Commercial projects tend to have larger number delta per commit as documented in, for example, [5]. Our results appear to support these earlier findings. We, therefore, propose:

Hypothesis 5.2: **HPC production projects will have more complex commits measured by the number of files modified in a commit than most open source projects.**

VI. Limitations

There are many limitations associated with empirical studies in general [24], and case studies [2] in particular. These include construct validity, internal validity, external validity and reliability. Furthermore, the use of operational data [6] adds additional complexities to validate issues related to the lack of context, missing information, or inaccurate records. Construct validity concerns how well variables reflect the constructs that they are supposed to operationalize, and how accurately the values of the variables are measured. We have consulted a variety of sources (to triangulate) to construct our measures as described above.

While they represent only a preliminary attempt to quantify such complex aspect as commit quality, they are fairly simple to obtain. Internal validity in case studies demonstrates that there is a chain of evidence from data collection to results. We have tried to carefully describe the steps that were taken to obtain the results. External validity establishes the domain to which a study's findings can be generalized. We have carefully selected HPC production projects and we expect that the findings would extend beyond the five analyzed projects, but, for example, the findings may not apply to other types of HPC projects. Reliability of a study relates to how much error is inherent in measuring outcomes and covariates. To assess that error we look at the outcome and predictor trends over time.

VII. Conclusions and Future Work

While the HPC community was one of the earliest to embrace code sharing, see, e.g., [25], the public sharing of version control tools and issue trackers that define open source development has lagged. Many of the HPC projects are using such tools at present, including the projects we study here. An open question is if the typical VCS and issue trackers are most suitable for HPC development practices and , if not, what modifications are needed to make HPC development most productive. We hope that our initial findings would help pose more precise questions in this area and the methods used would help answer such questions in the future.

Our case study goal was to investigate the difference between the development traces in HPC and other projects. The results of the study suggest the specific hypotheses that we plan to investigate on a more comprehensive set of projects. Inspired by reviewers comment we also plan to investigate how the "one-shot" research applications written with no maintenance in mind could benefit from the practices used in production codes. To accomplish that, we plan to obtain a sample of such research codes containing both, projects that have experienced a subsequent maintenance and projects that were abandoned after completion.

Acknowledgment

We would like to thank Chunyan Tang for her contributions to the early stages of this work during our course work. We are also very grateful to Geroge Bosilca, Stanimire Tomov, and Piotr Luszczek from Innovative Computing Laboratory (ICL) at the University of Tennessee, Knoxville who gave us access to BitBucket repos and commit comments history for the HPC projects we took investigate in our work and provided expert insights on development practices of production HPC codes.

References

[1] A. Mockus, "Software support tools and experimental work," in *Empirical Software Engineering Issues: Critical Assessments and Future Directions*, V. Basili and et al, Eds. Springer, 2007, vol. LNCS 4336, pp. 91–99. [Online]. Available: papers/SSTaEW.pdf

[2] R. K. Yin, *Case study research: Design and methods*. Sage publications, 2014.

[3] R. J. Hanson, F. T. Krogh, and C. L. Lawson, "Improving the efficiency of portable software for linear algebra," *ACM SIGNUM Newsletter*, vol. 8, no. 4, pp. 16–16, 1973.

[4] V. Basili, J. Carver, D. Cruzes, L. Hochstein, J. Hollingsworth, F. Shull, and M. Zelkowitz, "Understanding the high performance computing community: A software engineer's perspective," *IEEE Software*, vol. 25, no. 4, pp. 29–36, July/August 2008.

[5] A. Mockus, R. F. Fielding, and J. Herbsleb, "A case study of open source development: The Apache server," in *22nd International Conference on Software Engineering*, Limerick, Ireland, June 4-11 2000, pp. 263–272. [Online]. Available: http://dl.acm.org/authorize?2580

[6] A. Mockus, "Engineering big data solutions," in *ICSE'14 FOSE*, 2014. [Online]. Available: papers/BigData.pdf

[7] J. Xie, M. Zhou, and A. Mockus, "Impact of triage: a study of Mozilla and Gnome," in *ESEM '13*, 2013. [Online]. Available: papers/triage.pdf

[8] J. Xie, Q. Zhengand, M. Zhou, and A. Mockus, "Product assignment recommender," in *ICSE'14 Demonstrations*, 2014. [Online]. Available: http://dl.acm.org/authorize?6913517

[9] C. Thompson. (2014) 5 useful tips for a better commit message. [Online]. Available: http://robots.thoughtbot.com/5-useful-tips-for-a-better-commit-message

[10] S. Davies and M. Roper, "What's in a bug report?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 26:1–26:10. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652541

[11] OpenMPI. Openmpi project. [Online]. Available: http://icl.cs.utk.edu/open-mpi/

[12] OpenSHMEM. Openshmem project. [Online]. Available: http://icl.cs.utk.edu/graphics/posters/files/SC14-OpenSHMEM-UCCS.pdf

[13] PaRSEC. Parsec project. [Online]. Available: http://icl.cs.utk.edu/parsec/

[14] PLASMA. Plasma project. [Online]. Available: http://icl.cs.utk.edu/plasma/

[15] MAGMA. Magma project. [Online]. Available: http://icl.cs.utk.edu/magma/

[16] B. Tutorial. Bitbucket tutorial project. [Online]. Available: https://bitbucket.org/tutorials/tutorials.bitbucket.org

[17] Django-piston. Django-piston project. [Online]. Available: https://bitbucket.org/jespern/django-piston/wiki/Home

[18] L. Kernel. Linux kernel project. [Online]. Available: https://bitbucket.org/emiliolopez/linux

[19] T. Pope. A note about git commit messages. [Online]. Available: http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

[20] B.-E. Dahlberg. Writing good commit messages. [Online]. Available: https://github.com/erlang/otp/wiki/Writing-good-commit-messages

[21] StackOverflow. Standard to follow when writing git commit messages. [Online]. Available: http://stackoverflow.com/questions/15324900/standard-to-follow-when-writing-git-commit-messages

[22] C. Beams. How to write a git commit message. [Online]. Available: http://chris.beams.io/posts/git-commit/

[23] T. L. Graves and A. Mockus, "Inferring change effort from configuration management data," in *Metrics 98: Fifth International Symposium on Software Metrics*, Bethesda, Maryland, November 1998, pp. 267–273. [Online]. Available: papers/effort

[24] M. Jahoda, M. Deutsch, and S. W. Cook, "Research methods in social relations with special reference to prejudice. vol. 1, basic processes. vol. 2, selected techniques." 1951.

[25] S. Browne, J. Dongarra, E. Grosse, and T. Rowan, "The netlib mathematical software repository," *D-Lib Magazine, Sep*, 1995.