Improving HPC Development through Empirically Guided Analysis

A Proposal to:

Advancing University Research with High Performance Computing (HPC) through Increased Student Engagement

Philip Johnson
Collaborative Software Development Laboratory
Department of Information and Computer Sciences
University of Hawaii

Table of Contents

Application for Advancing University Research with High Performance Computing (HPC) through Increased Student Engagement

Instructions: Please complete the fields below. Mail, fax or email a signed and scanned copy as an attachment, to Dr. Susan T. Brown, 2532 Correa Rd., Building 37, Honolulu, HI 96822, stbrown@hawaii.edu, along with the 2-page project summary. (Summary may be an electronic attachment.)

Faculty Sponsor (Proposer):

Name:	**Philip Johnson**
Dept.	**Information and Computer Sciences**
Campus:	**Manoa**
Phone:	**808 956-3489**
Email:	**johnson@hawaii.edu**

Student:

Name: **Michael Paulding**

Undergraduate ___ Graduate _X__

Dept: **Information and Computer Sciences**

Campus: **Manoa**

Email: **mpauldin@hawaii.edu**

Academic Year 2005-6 _X___ Summer 2005____

Title of Project: **Improving HPC Development through Empirically Guided Analysis**

Amount Requested: **$20,928 ($18,198 salary + $2,730 fringe)**

Basis of amount requested for direct student support, with reference to University student employment guidelines and pay schedules (e.g., undergraduate student at step A4-1 with payrate X for Y hrs/wk for Z week): **Graduate research assistantship GA-5 for one year at 20 hours/wee**k

Description of any non-financial resources requested to execute the project (expected usage of HPC resources, software licenses, technical help): **Expected usage of HPC resources includes access to clusters for development of software and gathering of experimental data, and reasonable level of tech support for hardware/software issues encountered.**

How project will use the MHPCC resources:
Development of software and gathering of experimental data as described in the project proposal.

Signatures:

Faculty Sponsor – I agree to supervise this student in the execution of the proposed project proposed.

Signature:_________________________ Date:_____________

Student: I agree to work under the supervision of the faculty sponsor in the execution of the proposed project.

Signature:______________________ Date:_________________

Fiscal Officer - I have reviewed the amount requested and agree that it is the appropriate amount required for student employment according to standard University of Hawaii administrative procedures. I further agree to administer the student employment and any other activities proposed.

Signature:____________________________ Date:________________

Please refer any questions to Sue Brown at 956-2808 or email stbrown@hawaii.edu.

Summary of Project (maximum 2 pages) describing technical background, work to be done, relevant qualifications of the faculty sponsor and proposed student to execute the work, background on any additional assistance requested. Attachments may be included (student resume, faculty CV, related publications) but are not required.

Improving HPC Development Through Empirically Guided Analysis

Motivation

High performance computing systems are being applied to an increasingly wide variety of domains, including nuclear physics, crash simulation, fluid dynamics, climate modeling and bioinformatics. They are also becoming radically less expensive: the recently announced $1.5 million dollar BlueGene/L supercomputer will execute 24 teraflops per second. In comparison, the former fastest supercomputer, Japan's Earth Simulator, executes 36 teraflops per second but costs over $350 million dollars.

Unfortunately, these numbers do not tell the whole story. The DARPA High Productivity Computing Systems Program [1] and the Workshop on the Roadmap for the Revitalization of High-End Computing [2] note that dramatic increases in low-level HPC benchmarks of processor speed, memory access, and dollars/flop do not necessarily translate into increased development productivity. In other words, while the hardware is clearly getting faster and cheaper, the developer effort required to exploit these advances is becoming prohibitive. Software engineering, not hardware engineering, is becoming the limiting factor in the advance of HPC.

To address these issues, we have pursued collaborative research with SUN Microsystems and MHPCC with two broad goals: (1) The study of HPC development from a software engineering perspective, with the goal of identifying key HPC productivity bottlenecks and subsequent development of new tools or techniques to overcome them; and (2) The development of more comprehensive measures of HPC productivity for use in evaluating new and existing HPC architectures, which take into account not only the low-level hardware components, but the higher-level development costs associated with producing usable HPC applications using the architecture. Our work together so far has included the development of a framework for HPC productivity evaluation that uses "Purpose-Based Benchmarks" [3], and the organization of two workshops on HPC and Software Engineering [4].

Past Accomplishments

With the funding allocated to us during the academic year 2004-2005, we were pleased to accomplish all goals set forth in our application. Specifically, we achieved the following milestones through our research activities:

1. ***Implementation of the Truss Optimizer Benchmark on an MHPCC platform -*** For this research milestone, we developed a complete solution to the Truss Benchmark on the Squall system at MHPCC, making use of the IBM xlf compiler and MPI libraries provided in Squall's environment. The code was written in C++ and is portable between IBM and GNU compilers and UNIX environments. As anticipated, this benchmark provides an advantage over traditional ones, such as LINPACK, as it measures development activities, such as time required to implement the software, in addition to measuring runtime performance.

2. ***Acquisition of Truss Optimizer Benchmark measurements on Squall –*** In parallel with implementation, we were able to collect and analyze data for the Truss Benchmark project. Specifically, we were able gain traditional HPC measurements, such as speedup, and software engineering measurements, such as developer effort, code size and code complexity. We have also generated an informational website [6] documenting and interpreting our results, as well as providing open source code and data.

3. ***Extension of the Hackystat automated software measurement framework to support HPC measurement*** – For this milestone, we implemented a set of tools to collect measurement data particularly important to HPC development. One of the tools was a sensor to automatically collect information about code size and is able to distinguish between sequential and parallel code, such that their sizes could be measured independently. Another tool we created captured and sent C++ unit test data from its invocation to the Hackystat system. All of these tools require minimal developer interaction to install and configure and then collect data automatically with no direct developer intervention.

In addition to meeting these milestones, another rewarding aspect of our work was the acceptance of a paper [7] into the 2005 High-Performance Computer Architecture Conference in San Francisco and the presentation of this paper at the workshop for Productivity and Performance in High-End Computing held in conjunction with the conference. As a result of our participation in the conference, we were able to network with HPC professionals, promote the work performed and facilities offered at MHPCC to attendees and generate interest in our research.

Proposed Research

Now that we have direct and tangible results from developing the Truss Benchmark in an HPC environment on Squall and have created tools to collect and analyze HPC related data from our experience, we can apply and expand our tools and knowledge to projects at MHPCC and the broader HPC community.

The Information and Computer Sciences department has recently hired Dr. Henri Cassanova to teach software development and conduct research in the field of HPC. This provides a unique opportunity for collaboration between the University of Hawaii and MHPCC, such that students can utilize and promote MHPCC resources for their coursework and assignments and we can conduct empirical studies from a subject population local to the university. This opportunity enables us to apply our tools to other HPC projects, solicit feedback from a significant pool of HPC developers and create new tools or refine older ones to fully capture the HPC development process within the MHPCC environment.

In pursuit of these goals during the 2005-2006 academic year, the proposed funding will be used to support the following research activities.

1. **Implement a framework for evaluating HPC development approaches for MHPCC** – In order to understand HPC productivity, a comparison of the development approaches must be conducted. We propose to extend the Hackystat system to be able to evaluate multiple HPC packages, such as MPI, OpenMP, Co-Array Fortran and UPC. Specifically, we expect to be able to measure cost, in terms of development time, size, in terms of source lines of code and code expansion factor, as a ratio between size of a parallel solution and size of a serial solution.

2. **Enhance HPC tool support for the Hackystat system**– While implementing the Optimal Truss problem, we identified additional metrics and data that are important to HPC development. For example, when executing parallel code, there are many instances in which the code will fail on one or more processors, but not all. We propose to add a new tool to Hackystat to capture runtime data, especially failures, from applications such as mpirun and poe, and automatically classify the failures to assist developers in understanding them. These tools will run in the MHPCC environment.

3. **Categorical identification and classification of HPC bottlenecks** – There are many potential productivity bottlenecks in HPC, many of which constitute unnecessary repetitive actions. For example, frequent editing of IP addresses in machine files typically indicates that a developer is experiencing runtime problems with one or more processors. In our research, we propose to extend Hackystat to capture, classify and quantify bottlenecks, such as machine configuration in addition to many others.

4. **Collaboration between UH and MHPCC to conduct empirical studies of HPC development** – We are excited to collaborate with MHPCC and Dr. Cassanova to conduct empirical studies with students using HPC resources on Squall. Such experimentation is expected to yield empirical data for the HPC community as well as increased exposure to resources available at MHPCC.

At the conclusion of this year of research and development, we expect that the Hackystat/MHPCC technology infrastructure will be mature enough for general deployment and use, making MHPCC a leading center for empirical study of the software engineering of high performance computing applications.

References

[1] DARPA High Productivity Computing Systems Program, http://www.highproductivity.org/
[2] The Roadmap for the Revitalization of High-End Computing, http://www.cra.org/Activities/workshops/nitrd/
[3] S. Faulk, J. Gustafson, P. Johnson, A. Porter, W. Tichy, and L. Votta, *Measuring HPCS Productivity*, International Journal of High Performance Computing and Applications, vol. 18, no. 4, Winter 2004.
[4] International Workshop on Software Engineering for High Performance Computing System Applications, http://csdl.ics.hawaii.edu/se-hpcs/, 2004, 2005.
[5] Hackystat, A Framework for Automated Software Engineering Measurement, http://hackydev.ics.hawaii.edu
[6] Optimal Truss Problem, Specifications, Implementation and Insights, http://csdl.ics.hawaii.edu/Research/Truss
[7] P. Johnson, M. Paulding, *Understanding HPC Development through Automated Process and Product Measurement with Hackystat*, High Performance Computer Architecture Conference, February 2005.

Philip M. Johnson

Information and Computer Sciences
University of Hawaii
1680 East-West Road
Honolulu, HI 96822

(808) 956-3489
fax: (808) 956-3548
johnson@hawaii.edu
http://csdl.hawaii.edu/∼johnson/

Degrees

Ph.D. in Computer Science, University of Massachusetts, Amherst. 1990

M.S. in Computer Science, University of Massachusetts, Amherst. 1985

B.S. in Computer Science, University of Michigan, Ann Arbor. 1980

B.S. in Biology, University of Michigan, Ann Arbor. 1980

Research and Teaching Experience

Professor 2001—present
Associate Professor 1995—2001
Assistant Professor 1990—1995
Department of Information and Computer Sciences, University of Hawaii.

- Director, Collaborative Software Development Laboratory
- Graduate Chair, Information and Computer Sciences, 1998-2001

Senior Research Fellow 1997
Distributed Systems Technology Centre, University of Queensland, Brisbane, Australia.

- Research on computer-supported cooperative work technologies

Research Assistant 1984—1986, 1987—1990
Department of Computer Science, University of Massachusetts.

- Ph.D. thesis on structural evolution in software development
- Research on automated Ada package restructuring
- Research on natural language processing tools.

Teaching Assistant 1983—1984
Department of Computer Science, University of Massachusetts.

- Coursework support in software engineering.

Lecturer 1981—1982
Department of Computer and Communication Sciences, University of Michigan.

- Taught introductory programming course.

Industry Experience

Member, Board of Directors 2003-present
Tiki Technologies, Inc., Honolulu, Hawaii.

- Tiki Technologies develops internet software including spam detection systems.

Member, Board of Directors 2002-present
Lavanet, Inc., Honolulu, Hawaii.

- Lavanet is an Internet Service Provider, Network Engineering, and Web Development Services company.

Member, Board of Directors 1999-present
Hawaii Strategic Development Corporation, Honolulu, Hawaii.

- HSDC is a State-sponsored organization that works to support the growth of the venture capital industry in Hawaii.

Member, Professional Advisory Board 2000-present
BreastCancer.org, Philadelphia, PA.

- BreastCancer.org is a non-profit organization dedicated to helping those living with breast cancer.

Member, Board of Directors 2000-2004
High Technology Development Corporation, Honolulu, Hawaii.

- HTDC is a State-sponsored organization whose mission is to support the growth of the high technology industry in Hawaii.

Co-Founder 2000
hotU, Inc., Honolulu, Hawaii.

- Served as interim Chief Technology Officer during initial formation of company to provide Internet-based services to college student market. Currently serve on Technology Advisory Board.

Consulting Software Engineer 1994—present
Honolulu, Hawaii.

- Providing project management and software engineering services to local and national companies.

Member, Professional Advisory Board 2000-2002
Referentia, Inc., Honolulu, HI.

- Referentia develops E-learning multimedia packages.

Programmer 1986—1987
Department of Computer Science, University of Massachusetts.

- Developed control shell for GBB, an AI knowledge base and inference environment.

Systems Programmer 1982—1983
Software Services Corporation, Ann Arbor, MI.

- Developed software quality assurance and validation tools for Ford Motor Company.

Systems Analyst 1981—1983

Veterans Hospital, Ann Arbor, MI.

- Developed real-time data acquisition and signal processing software to control hardware for psychophysiological experimentation.

Programmer 1978
Great Lakes Software Systems, Ann Arbor, MI.

- Implemented an accounts receivable package in COBOL.

Journal Publications

P. M. Johnson and H. Kou and M. Paulding and Q. Zhang and and A. Kagawa and T. Yamashita, *Improving software development management through software project telemetry*, To appear in IEEE Software, 2005.

S. Faulk and J. Gustafson and P. Johnson and A. Porter and W. Tichy and L. Votta, *Measuring HPC Productivity*, International Journal of High Performance Computing Applications, December 2004.

P. M. Johnson and M. L. Moffett and B. T. Pentland, *Lessons learned from VCommerce: A virtual environment for interdisciplinary learning about software entrepreneurship*. Communications of the ACM, Vol. 46, No. 12, December, 2003.

P. M. Johnson and C. A. Moore and J. A. Dane and R. S. Brewer, *Empirically Guided Software Effort Guesstimation*. IEEE Software, Vol. 17, No. 6, December 2000.

P. M. Johnson and A. M. Disney, *A Critical Analysis of PSP Data Quality: Results from a Case Study*. Journal of Empirical Software Engineering, Volume 4, December, 1999.

P. M. Johnson and A. M. Disney, *The Personal Software Process: A Cautionary Case Study*. In IEEE Software, Volume 15, No. 6, November, 1998.

P. M. Johnson, *Reengineering Inspection*. In Communications of the ACM, Volume 41, No. 2, February, 1998.

P. M. Johnson and D. Tjahjono, *Does Every Inspection Really Need A Meeting?*, In Journal of Empirical Software Engineering, Volume 4, No. 1, January 1998.

A. A. Porter and P. M. Johnson, *Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies*. In IEEE Transactions on Software Engineering, vol. 23, no. 3, March 1997.

P. M. Johnson, *Design for Instrumentation: High Quality Measurement of Formal Technical Review*. Software Quality Journal, Volume 5, March, 1996.

D. Wan and P. M. Johnson, *Experiences with CLARE: a Computer-Supported Collaborative Learning Environment*. In the International Journal of Human-Computer Studies, Volume 41, December, 1994.

P. M. Johnson, *Experiences with EGRET: An Exploratory Group Work Environment*. In Collaborative Computing 1(1), March, 1994.

B. Walker, M. Walker, S. Achem, P. Johnson, and R. Gregg. *The Clinical Significance of Electrogastrography*. In Psychophysiology 20: 1983.

Book Chapters

P. M. Johnson, *An Instrumented Approach to Improving Software Quality through Formal Technical Review*. In Software Inspection: An Industry Best Practice. David A. Wheeler, Bill Brykczynski, and Reginald N. Meeson, Jr., Editors. IEEE Computer Society Press. 1996

Also appearing in the Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy. 1994.

P. M. Johnson and W. G. Lehnert, *Beyond Exploratory Programming: A Methodology and Environment for Natural Language Processing.* In Artificial Intelligence and Software Engineering, D. Partridge, editor. Ablex, 1990.

Also appearing in Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86), Philadelphia, PA.

D. Corkill, K. Gallagher, and P. M. Johnson, *Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures*. In Readings in Distributed Artificial Intelligence, A. Bond and L. Gasser, editors. Morgan-Kaufman, 1988.

Also appearing in Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), Seattle, WA.

Conference Publications

P. M. Johnson, H. Kou, J. Agustin, Q. Zhang, A. Kagawa, T. Yamashita, *Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat-UH*, In Proceedings of the 2004 Symposium on Empirical Software Engineering, Los Angeles, CA., August 2004.

P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, and W. Doane, *Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined*, In Proceedings of the 2003 International Conference on Software Engineering, Portland, OR., May, 2003.

P. M. Johnson, *Leap: A "Personal Information Environment" for Software Engineers*. In Proceedings of the 1999 International Conference on Software Engineering, Los Angeles, CA., May 1999.

A. M. Disney, P. M. Johnson, *Investigating Data Quality Problems in the PSP*. In Proceedings of the Sixth International Symposium on the Foundations of Software Engineering, Orlando, FL., November, 1998.

P. M. Johnson, D. Tjahjono, *Assessing software review meetings: A controlled experimental study using CSRS*. In Proceedings of the 1997 International Conference on Software Engineering, Boston, MA., May 1997.

D. Wan and P. M. Johnson, *Computer Supported Collaborative Learning using CLARE: the Approach and Experimental Findings*. In Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, Chapel Hill, NC. 1994.

P. M. Johnson, *Supporting Technology Transfer of Formal Technical Review through a Computer Supported Collaborative Review System*. In Proceedings of the Fourth International Conference on Software Quality, Reston, VA. 1994

P. M. Johnson, *An Instrumented Approach to Improving Software Quality through Formal Technical Review*. In Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy. 1994.

P. M. Johnson, D. Tjahjono, D. Wan, R. Brewer, *Experiences with CSRS: An Instrumented Software Review Environment*. In Proceedings of the 11th Annual Pacific Northwest Software Quality Conference, Portland, OR. 1993.

P. M. Johnson, D. Tjahjono, *Improving Software Quality through Computer Supported Collaborative Review*. In Proceedings of the Third European Conference on Computer Supported Cooperative Work, Milan, Italy. 1993.

P. M. Johnson, *Supporting Exploratory CSCW with the EGRET Framework*. In Proceedings of the ACM 1992 Conference on Computer Supported Cooperative Work, Toronto, Canada. 1992.

P. M. Johnson, D. Hildum, A. Kaplan, C. Kay, and J. Wileden, *An Ada Restructuring Assistant*. In Proceedings of the Fourth Annual Conference on Artificial Intelligence and Ada, Fairfax, VA. 1988.

D. Corkill, K. Gallagher, and P. M. Johnson, *Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures*. In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), Seattle, WA.

P. M. Johnson and W. G. Lehnert, *Beyond Exploratory Programming: A Methodology and Environment for Natural Language Processing*. In Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86), Philadelphia, PA.

Workshop Publications

P. M. Johnson, M. G. Paulding, *Understanding HPC Development through Automated Process and Product Measurement with Hackystat*, Proceedings of the Second Workshop on Productivity and Performance in High-End Computing (P-PHEC), February, 2005.

P. M. Johnson, *You can't even ask them to push a button: Toward ubiquitous, developer-centric, empirical software engineering*, Proceedings of the Workshop on New Visions for Software Design and Productivity: Research and Applications, December, 2001.

P. M. Johnson, *Project LEAP, Lightweight, Empirical, Anti-measurement dysfunction, and Portable Software Developer Improvement*, Software Engineering Notes, Volume 24, Number 6, December 1999.

P. M. Johnson, *Egret: A Framework for Advanced CSCW Applications*, Software Engineering Notes, Volume 21, Number 5, September 1996.

P. M. Johnson, *Assessing software review meetings: An empirical study using CSRS*, Appearing in the 1996 International Software Engineering Research Network Meeting (ISERN'96), Sydney, Australia, August, 1996.

P. M. Johnson and Carleton Moore, *Investigating Strong Collaboration with the Annotated Egret Navigator*, Appearing in the Fourth IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 95), April, 1995.

P. M. Johnson, *Computer Supported Formal Technical Review with CSRS*, Software Inspection and Review Organization Newsletter, Volume 5, Number 3, December, 1994.

P. M. Johnson, *Collaboration-in-the-large vs. Collaboration-in-the-small*. Appearing in Proceedings of the 1994 CSCW Workshop on Software Architectures for Cooperative Systems, Chapel Hill, VA. October, 1994.

P. M. Johnson, *From Principle-centered to Organization-centered Design: A Case Study of Evolution in a Computer-Supported Formal Technical Review Environment*. Appearing in the Proceedings of the 15th Interdisciplinary Workshop on Informatics and Psychology, Scharding, Austria, 1994.

P. M. Johnson, *Report from the 1993 ECSCW Workshop on Tools and Technologies.* SIGOIS Bulletin, April, 1994.

P. M. Johnson, *Methodological Issues in CSCW Research.* Position paper for the 1993 European Conference on Computer Supported Cooperative Work, Workshop on Tools and Technologies, Milan, Italy. 1993.

P. M. Johnson, *An Architectural Perspective on EGRET*. In Proceedings of the ACM 1992 Conference on Computer Supported Cooperative Work, Workshop on Tools and Technologies, Toronto, Canada. 1992.

P. M. Johnson, *Collaborative Software Review for Capturing Design Rationale*. In Proceedings of the 1992 AAAI Workshop on AI and Design Rationale, San Jose, CA. 1992.

D. Wan and P. M. Johnson, *Supporting Scientific Learning and Research Review using COREVIEW*. In Proceedings of the 1992 AAAI Workshop on Communicating Scientific and Technical Knowledge, San Jose, CA. 1992.

P. M. Johnson, *EGRET: Exploring Open, Evolutionary, and Emergent Collaborative Systems.* In Proceedings of the 1991 European Conference on Computer Supported Cooperative Work, Tools and Technologies Workshop, Amsterdam, The Netherlands. 1991.

P. M. Johnson, *Structural Evolution in Exploratory Software Development.* In Proceedings of the 1989 AAAI Spring Symposium on AI and Software Engineering, Stanford University, CA. 1989.

S. Founds and P. M. Johnson, *A Knowledge-based Rhythm Composition Tool*. In Proceedings of the 1989 IJCAI Workshop on Artificial Intelligence and Music, Detroit, MI. 1989.

P. M. Johnson, *Integrating BB1-style Control into the Generic Blackboard System*. In Proceedings of the 1987 AAAI Workshop on Blackboard Systems, Seattle, WA. 1987.

P. M. Johnson, *Combining Software Engineering and Artificial Intelligence*. In Proceedings of the First International Workshop on Computer-Aided Software Engineering, Cambridge, MA. 1987.

D. Corkill, K. Gallagher, and P. M. Johnson, *From Prototype to Product: Evolutionary Development from within the Blackboard Paradigm*. In Proceedings of the Workshop on High-level Tools for Knowledge-based Systems, Columbus, OH. 1986.

P. M. Johnson, *Requirements Definition for a PLUMber's Apprentice*. In Proceedings of the Second Annual Workshop on Theoretical Issues in Conceptual Information Processing, New Haven, CT. 1985.

Tutorial Presentations

Java: What's it all about? Half-day tutorial presented at the Pacific New Media Center, Honolulu, HI, November, 1997.

The TekInspect Software Review Method: Reviewer Training and *The TekInspect Software Review Method: Moderator Training*. Two day tutorial presented at Tektronix, Inc, Beaverton, OR. July, 1996.

Inspection Quick Start: An accelerated, pragmatic introduction for software engineers and managers. Full-day tutorial presented at Tektronix, Inc, Beaverton, OR. May, 1996.

Improved Formal Technical Reviews: Beyond Fagan Code Inspections. Half-day tutorial presented at the 17th International Conference on Software Engineering, Seattle, WA. April, 1995.

Formal Technical Review: Theory and Practice — Past, Present, and Future. Half-day tutorial presented at the Fourth International Conference on Software Quality, Reston, VA. 1994

UNIX and Trends in Next Generation Operating Systems. Half-day tutorial presented at the Japan-America Institute for Management Science. Honolulu, HI. 1992, 1993, 1994.

Invited Talks

Tool Support for Experimentation. Panel chair and member, presented at the 2003 meeting of the International Software Engineering Research Network, Rome, Italy, 2004.

e-World Experimentation. Panel chair and member, presented at the 2001 meeting of the International Software Engineering Research Network, Strathclyde, Scotland, August, 2001.

Experimental Software Engineering in Internet Startups: An oxymoron? Talk presented at the 2000 meeting of the International Software Engineering Research Network, Honolulu, HI. October, 2000.

Process improvement is dead! Long live the rising tide! Talk presented at Microsoft Corporation, Seattle, Washington. November, 1999.

Questioning assumptions in the PSP, Part II. Talk presented at the 1999 meeting of the International Software Engineering Research Network, Oulu, Finland. June, 1999.

Introduction to Software Engineering in the Collaborative Software Development Laboratory. Talk presented at "A presentation of leading-edge scientific projects and programs for the Honorable Benjamin J. Cayetano, Governor, State of Hawaii". University of Hawaii, April, 1999.

Investigating Data Quality Problems in the PSP. Talk presented at the Sixth International Symposium on the Foundations of Software Engineering, Orlando, FL, November, 1998.

From the PSP to Project LEAP. Talk presented at the 1998 Meeting of the International Software Engineering Research Network, Naperville, IL, October, 1998.

What's Quality Got To Do With It? Talk presented at the School of Information Technology, University of Queensland, St. Lucia, Australia, July, 1997.

Java: What's so Special? Talk presented at the Annual Meeting of the Data Processing Management Association, Western Region. Honolulu, HI, September, 1996.

Object Oriented Programming: From Scandinavia to the South Pacific, Simula to OO Cobol. Talk presented to the Data Processing Management Association, Honolulu Chapter, April 1996.

Reengineering Inspection: The Future of Formal Technical Review. Talk presented at: Tektronix, Inc, Beaverton, OR; Andersen Consulting, Chicago, IL; Motorola, Inc, Austin, TX. 1996.

Supporting Software Quality through Computer Supported Formal Technical Review. Invited talk for the Distinguished Software Scientists Visiting Speaker Program, Tektronix, Inc. Beaverton, OR. 1994.

Supporting Technology Transfer of Formal Technical Review through a Computer Supported Collaborative Review System. Talk presented at the Fourth International Conference on Software Quality, Reston, VA. 1994

Lessons Learned from Designing Computer-mediated Collaboration for Formal Technical Review. Talk presented at the 15th Interdisciplinary Workshop on Informatics and Psychology, Schärding, Austria, 1994.

An Instrumented Approach to Improving Software Quality through Formal Technical Review. Talk presented at: the 16th International Conference on Software Engineering, Sorrento, Italy; the University of Aalborg, Denmark. 1994.

Experiences with CSRS: An Instrumented Software Review Environment. Talk presented at the Eleventh Annual Pacific Northwest Quality Conference, Portland, OR. October, 1993.

Improving Software Quality through Computer-Supported Formal Technical Review. Invited talk presented at the Naval Command, Control, and Ocean Surveillance Center, San Diego, CA. October, 1993.

Improving Software Quality through Computer-Supported Collaborative Review. Talk presented at the 1993 European Conference on Computer Supported Cooperative Work. Milan, Italy. 1993.

Why CSCW Developers are Bad at CSCW Research. Keynote address presented at the 1993 European Conference on Computer Supported Cooperative Work, Tools and Technologies Workshop. Milan, Italy. 1993.

CSRS: A Collaborative Software Review Environment. Poster session at the 15th International Conference on Software Engineering. Baltimore, MD. 1993.

Working Notes on Collaborative Design of a Collaborative Spreadsheet. Talk presented at the 1992 International Conference on Computer Supported Cooperative Work, Workshop on Tools and Technologies. Toronto, Canada. 1992.

Supporting Exploratory CSCW with the Egret Framework. Talk presented at the 1992 International Conference on Computer Supported Cooperative Work. Toronto, Canada. 1992.

Coordination in the Egret Framework. Talk presented at the 1992 AAAI Workshop on AI and Design Rationale. San Jose, CA. 1992.

International Aspects of Computer Supported Cooperative Work. Talk presented at the Hawaii International Software Conference. Honolulu, HI. 1991

Type Flow Analysis for Exploratory Software Development. Talk presented at: MITRE Corporation, University of South Carolina at Columbia, Siemens Corporation, University of California at San Diego, MCC, Portland State University, University of Hawaii at Manoa, and the University of Massachusetts at Amherst. 1990.

An Ada Restructuring Assistant. Talk presented at the Fourth Annual Conference on Artificial Intelligence and Ada. Fairfax, VA. 1988.

Integrating BB1-Style Control into the Generic Blackboard System. Talk presented at the 1987 AAAI Workshop on Blackboard Systems. Seattle, WA. 1987.

How can Knowledge-based Approaches Help Computer-Aided Software Engineering? Panel member at the First International Workshop on Computer-Aided Software Engineering. Cambridge, MA. 1987.

Combining Software Engineering and Artificial Intelligence. Talk presented at the First International Workshop on Computer-Aided Software Engineering. Cambridge, MA. 1987.

Beyond Exploratory Programming: A Methodology and Environment for Natural Language Processing. Talk presented at AAAI 1986. Philadelphia, PA. 1986.

An Introduction to the Plumber's Apprentice. Talk presented at General Electric Corporation Research and Development. Schenectady, NY. 1986.

Requirements Definition for a Plumber's Apprentice. Talk presented at the 2nd Annual Workshop on Theoretical Issues in Conceptual Information Processing. New Haven, CT. 1985.

Computers in Psychophysiology: Hardware and Software Considerations. Lecture series presented at the Veterans Hospital, Ann Arbor, MI. 1982.

Awarded Grant Support

Student Engagement Grant, P. M. Johnson, Principal Investigator. University of Hawaii and Maui High Performance Computing Center. $20,928. 2004.

Eclipse Innovation Grant Award, P. M. Johnson, Principal Investigator. IBM Corporation. $15,000. 2004.

Supporting development of highly dependable software through continuous, automated, in-process, and individualized software measurement validation. P. M. Johnson, Principal Investigator. Joint NSF/NASA Highly Dependable Computing Program. $638,000. 2002-2006.

Aligning the financial services, fulfillment distribution infrastructure, and small business sectors in Hawaii through B2B technology innovation. P. M. Johnson, Principal Investigator. University of Hawaii New Economy Research Grant Program. $30,000. 2000-2001.

Internet Entrepreneurship: Theory and Practice. P. M. Johnson and Glen Taylor, Principal Investigators. University of Hawaii Entrepreneurship Course Development Grant. $10,000. 1999-2000.

Java-based software engineering technology for high quality development in "Internet Time" organizations. P. M. Johnson, Principal Investigator. Sun Microsystems Academic Equipment Grant Program. $39,205. 1999.

Project LEAP: Lightweight, Empirical, Anti-measurement dysfunction, and Portable Software Developer Improvement. P. M. Johnson, Principal Investigator. National Science Foundation. $265,000. 1998-2001.

Internet-enabled Engineering Tool for Dynamically Analyzing and Planning World-Wide Subsea Cable and Array Installations. P. M. Johnson, Principal Investigator. Makai Ocean Engineering, Inc. $83,286. 1998-1999.

Kona: A distributed, collaborative technical review environment. P. M. Johnson, Principal Investigator. Digital Equipment Corporation External Research Program. $101,413. 1997.

Collaborative Software Development Laboratory Industrial Affiliates Program: Makai Ocean Engineering, Inc., P. M. Johnson, Principal Investigator. $10,000. 1997.

Collaborative Software Development Laboratory Industrial Affiliates Program: Tektronix, Inc., P. M. Johnson, Principal Investigator. $45,000. 1996-1998.

Improving Software Quality through Instrumented Formal Technical Review, P. M. Johnson, Principal Investigator. National Science Foundation. $161,754. 1995-1997.

Collaboration Mechanisms for Project HI-TIME: Hawaii Telecommunications Infrastructure Modernization and Expansion: A Model for Statewide Strategic Planning, P. M. Johnson, Principal Investigator. Subcontract with the Pacific International Center for High Technology Research. $30,280. 1995

Three Dimensional Interfaces for Evolving Collaborative Systems. P. Johnson, Principal Investigator. University of Hawaii Research Council Seed Money Grant, $5,000. 1992-1993.

Support for Structural Evolution in Exploratory Software Development. P. M. Johnson, Principal Investigator. National Science Foundation Research Initiation Award Program in Software Engineering. $54,810. 1991-1993.

An Investigation of Software Structure Evolution. P. M. Johnson, Principal Investigator. University of Hawaii Research Council Seed Money Grant, $6,000. 1990-1991.

Professional Activities

Editorial Board, Journal of Empirical Software Engineering, 2004-present.

Program Chair, Second International Workshop on Software Engineering for High Performance Computing System Applications, St. Louis, MO, May, 2005.

Program Committee Member, PROFES 2005, Oulu, Finland, June 2005.

Program Chair, First International Workshop on Software Engineering for High Performance Computing System Applications, Edinburgh, Scotland, May, 2004.

Program Committee Member, XP/Agile Universe, Calgary, CA, August 2004.

Program Committee Member, International Software Metrics Symposium, 2003-2004.

Program Committee Member, International Symposium on Empirical Software Engineering, 2002-2004.

Editorial Board, IEEE Transactions on Software Engineering, 2000-2004.

Program Chair, International Software Engineering Research Network Annual Meeting, Honolulu, HI, 2000.

Member, State of Hawaii Millenium Workforce Development Initiative, 1999.

Program Committee Member, European Conference on Computer Supported Cooperative Work, Copenhagen, Denmark, 1999.

Judge, Hawaii State Science Fair, Honolulu, Hawaii, 1998-present.

Founder and Chair, Hawaii Java Users Group, Honolulu, Hawaii, 1996-present.

Member, International Software Engineering Research Network (ISERN), 1996-present.

Program Committee Member, European Conference on Computer Supported Cooperative Work, Lancaster, England, 1997.

Advisory Board Member, The International Journal of Computer Supported Cooperative Work, 1997-2004.

Editor, the WWW Formal Technical Review Archive.
http://www.ics.hawaii.edu/∼johnson/FTR/.

Editor, the WWW Software Inspection and Review Organization (SIRO) Home Page.
http://www.ics.hawaii.edu/∼siro/.

Program Organizer, Software Architectures for Cooperative Systems Workshop, 1994 ACM Conference on Computer Supported Cooperative Work, Chapel Hill, North Carolina.

Program Chair, CSCW Tools and Technologies Workshop, 1993 European Conference on Computer Supported Cooperative Work, Milan, Italy.

Program Organizer, CSCW Tools and Technologies Workshop, 1992 ACM Conference on Computer Supported Cooperative Work, Toronto, Canada.

Reviewer, IEEE Transactions on Software Engineering, IEEE Software, ACM Transactions on Software Engineering and Methodology, Hawaii International Conference on System Sciences, Sixth International Conference on Computing and Information, IEEE Computer, the 1993 Conference on Organizational Computing Systems, the 1993 International Conference on Computer Applications in Industry and Engineering, the Journal of Collaborative Computing, Artificial Intelligence in Engineering, Design, and Manufacturing (AI-EDAM), ACM Transactions on Programming Languages and Systems, The AI Handbook, Volume 4 (Chapter on AI and Software Engineering), the 1991 Conference on Software Maintenance.

Awards and Honors

Coach of the Year, American Youth Soccer Association, Region 100, 2001.

Honorary member, Golden Key International Honour Society, 2001.

University of Hawaii Presidential Citation for Meritorious Teaching, 1994.

Computer and Information Science Department Fellowship, 1989-1990.

University of Massachusetts Graduate School Fellowship, 1985-1986.

Michael George Paulding

mpauldin@hawaii.edu

Present Address
2349C Palolo Ave
Honolulu, HI 96816
(914) 815-2921 - cell

Permanent Address
15 Quincy Lane
White Plains, NY 10605
(914) 948-5489

EDUCATION:

University of Hawaii at Manoa, Honolulu, HI
Degree: Candidate for Ph.D. in Information and Computer Sciences
Cumulative G.P.A. – 3.96/4.0

Bucknell University, Lewisburg, PA
Degree: BS in Computer Science and Engineering, Magna Cum Laude, May 2002

Computer Experience: Sun Certified Java Programmer, C/C++, Visual Basic, Perl, Eiffel, LISP, Scheme, Pascal, MIPS, Sun Solaris, Linux and UNIX Workstations, Windows XP/ NT, TCP/IP, MS Office Suite

WORK EXPERIENCE:

Research Assistant, Collaborative Software Development Lab, UHM (01/2004 – Present)
- Performing research on measurement and analysis of software telemetry data to improve productivity of high performance and parallel computing systems.

Graduate Intern, Sun Microsystems, Inc. (Summer 2004)
- Implemented purpose-based benchmarks to identify key High Performance Computing System (HPCS) productivity bottlenecks and developed tools and techniques to overcome them.

Professional, JP Morgan Chase (08/2002 – 08/2003)
- Developed a real-time application to support in-house traders calculate profit maximizing prices from fixed income brokers.
- Designed a distributed platform to compute financial analytics for bonds and other fixed income instruments, processed in real-time.

Applications Delivery Intern, JP Morgan Chase (05/2001 – 08/2001)
- Developed VB application to verify fixed income broker pages between trader monitoring software and a remote database. Automated log to report discrepancies.
- Converted existing trader application to retrieve real time prices from a Tib publishing backbone. Modified trader application for improved performance.

TEACHING EXPERIENCE:

Lecturer, Kapi'olani Community College (01/2005 – Present)
- Co-taught ICS111 - Introduction to Computer Science I with Java
- Course was delivered through face-to-face and online meetings.

Teaching Assistant, Programming Languages, Networks and Web Programming courses, University of Hawaii at Manoa (08/2003 – 05/2004)
- Provided face-to-face information and remedial sessions for students outside scheduled course hours.
- Designed grading schema and assessed student work for assignments and exams.

Calculus I, II, III Tutor, Bucknell University (01/2000 – 05/2002)
- Met biweekly with two students in need of Calculus help throughout semesters.
- Underwent extensive training through the Career Development Center for improved communication in mathematics.

ACTIVITIES / : HONORS

"A Bounded Linear Approximation for Multiprocessor Job Scheduling" - Research with Dr. Antonio Miranda, Bucknell University (11/2001-05/2002)
Tau Beta Pi, National Engineering Honor Society (10/2001-Present)
Alpha Lambda Delta, National Honor Society (05/1999-Present)
Phi Eta Sigma, National Academic Fraternity (08/1999-Present)

Understanding HPC Development through Automated Process and Product Measurement with Hackystat

Philip M. Johnson
Michael G. Paulding
Collaborative Software Development Laboratory
Department of Information and Computer Sciences
University of Hawai'i
Honolulu, HI 96822
johnson@hawaii.edu
mpauldin@hawaii.edu

Abstract

The high performance computing (HPC) community is increasingly aware that traditional low-level, execution-time measures for assessing high-end computers, such as flops/second, are not adequate for understanding the actual productivity of such systems. In response, researchers and practitioners are exploring new measures and assessment procedures that take a more wholistic approach to high performance productivity. In this paper, we present an approach to understanding and assessing development-time aspects of HPC productivity. It involves the use of Hackystat for automatic, non-intrusive collection and analysis of six measures: Active Time, Most Active File, Command Line Invocations, Parallel and Serial Lines of Code, Milestone Test Success, and Performance. We illustrate the use and interpretation of these measures through a case study of small-scale HPC software development. Our results show that these measures provide useful insight into development-time productivity issues, and suggest promising additions to and enhancements of the existing measures.

1. Introduction

High performance computing systems are becoming mainstream due to decreasing costs and increasing numbers of application areas with computation and/or data intensive processing. With this interest, however, comes new challenges. For example, recent initiatives in the HPC community [8, 1] have concluded that low-level HPC benchmarks of processor speed and memory access times no longer necessarily translate into high-level increases in actual development productivity. Put another way, the bottleneck in high performance computing systems is increasingly due to software engineering, not hardware engineering.

To make matters even more interesting, high performance computing application development often differs in significant ways from the systems and development processes traditionally addressed by the software engineering community:

- The requirements often include conformance to sophisticated mathematical models. Indeed, requirements may often take the form of an executable model in a system such as Mathematica, and the implementation involves porting to the HPC system.

- The software development process, or "workflow" for HPC application development may differ profoundly from traditional software engineering processes. For example, one scientific computing workflow, dubbed the "lone researcher", involves a single scientist developing a system to test a hypothesis. Once the system runs correctly once and returns its results, the scientist has no further need of the system. This contrasts with standard software engineering lifecycle models, in which the useful life of the software is expected to begin, not end, after the first correct execution.

- "Usability" in the context of HPC application development may revolve around optimization to the machine architecture so that computations complete in a reasonable amount of time. The effort and resources involved in such optimization may exceed initial development of the algorithm.

Fortunately, there is an emerging interdisciplinary community involving both HPC and software engineering researchers and practitioners who are attempting to define new ways of measuring high performance computing systems, ways which take into account not only the low-level hardware components, but also the higher-level productivity costs associated with producing usable HPC applications.

This paper presents an approach to investigating the software engineering problems associated with high performance computing system application development. It involves the introduction of technology into the HPC development environment which unobtrusively gathers process and product data. This process and product data can be used for two purposes. First, it can be used to provide a more wholistic perspective on productivity, one that includes measures of performance, functionality, and development. Second, it can be used to provide new insight into the process of high performance system application development, which can be used to identify bottlenecks in the development process and assess the consequences of process or product changes on these bottlenecks. We have been applying this approach to an ongoing case study of high performance computing system application development in our laboratory, and this paper reports on our initial results.

The remainder of the paper is organized as follows. Section 2 introduces the technology we have developed, called Hackystat, which supports unobtrusive collection and analysis of product and process measures. Section 3 introduces "Software Project Telemetry", which is the principal approach to measurement collection and interpretation we have adopted for this research. Section 4 introduces a case study adapted from the Truss Purpose-based Benchmark (PBB) [3], which uses the problem specification but collects and analyzes an alternative set of metrics. Section 5 presents our initial conclusions from the use of these metrics and our future directions.

2. Automated process and product measurement with Hackystat

An important characteristic of our approach to understanding HPC software development and productivity is that measures of product and process must be automatically collected. This requirement limits the kinds of data we can collect, but dramatically lowers the cost of collecting these measures and provides a level of scalability for measurement not possible with expensive, manual data collection.

For the past several years, we have been developing a framework for automated software development process and product metric collection and analysis called Hackystat. This framework differs from other approaches to automated support for product and process measurement in one or more of the following ways:

- Hackystat uses sensors to unobtrusively collect data from development environment tools; there is no chronic overhead on developers to collect product and process data.
- Hackystat is tool, environment, process, and application agnostic. The architecture does not suppose a specific operating system platform, a specific integrated development environment, a specific software process, or specific application area. A Hackystat system is configured from a set of modules that determine what tools are supported, what data is collected, and what analyses are run on this data.
- Hackystat is intended to provide in-process project management support. Many traditional software metrics approaches are based upon the "project repository" method, in which data from prior completed projects are used to make predictions about or support control of a current project. In contrast, Hackystat is designed to collect data from a current, ongoing project, and use that data as feedback into the current project.
- Hackystat provides infrastructure for empirical experimentation. For those wishing to compare alternative approaches to development, or for those wishing to do longitudinal studies over time, Hackystat can provide a low-cost approach to gathering certain forms of project data.
- Hackystat is open source and is available to the academic and commercial software development community for no charge.

The design of Hackystat [6] has resulted from of prior research in our lab on software measurement, beginning with research into data quality problems with the PSP [5] and which continued with the LEAP system for lightweight, empirical, anti-measurement dysfunction, and portable software measurement [7].

To use Hackystat, the project development environment is instrumented by installing Hackystat sensors, which developers attach to the various tools such as their editor, build system, configuration management system, and so forth. Once installed, the Hackystat sensors unobtrusively monitor development activities and send process and product data to a centralized web service. If a user is working offline, sensor data is written to a local log file to be sent when connectivity can be established with the centralized web service. Project members can then log in to the web server to see the collected raw data and run analyses that integrate and abstract the raw sensor data streams into telemetry. Hackystat also allows project members to configure "alerts" that watch for specific conditions in the sensor data stream and send email when these conditions occur.

Hackystat is an open source project with sources, binaries, and documentation available at http://www.hackystat.org. There is also a public server available at http://hackystat.ics.hawaii.edu. Hackystat has been under development for approximately three years, and currently consists of around 900 classes and 60,000 lines of code. Sensors are available for a variety of tools including Eclipse, Emacs, JBuilder, Jupiter, Jira, Visual Studio, Ant, JUnit, JBlanket, CCCC, DependencyFinder, Harvest, LOCC, Office, and CVS.

3. Software Project Telemetry

A major application of Hackystat has been the development of a new approach to software measurement analysis called "Software Project Telemetry". We define Software Project Telemetry as a style of software engineering process and product collection and analysis which satisfies the following properties:

Software project telemetry data is collected automatically by tools that unobtrusively monitor some form of state in the project development environment. In other words, the software developers are working in a "remote or inaccessable location" from the perspective of metrics collection activities. This contrasts with software metrics data that requires human intervention or developer effort to collect, such as PSP/TSP metrics [4].

Software project telemetry data consists of a stream of time-stamped events, where the time-stamp is significant for analysis. Software project telemetry data is thus focused on evolutionary processes in development. This contrasts, for example, with Cocomo [2], where the time at which the calibration data was collected about the project is not significant.

Software project telemetry data is continuously and immediately available to both developers and managers. Telemetry data is not hidden away in some obscure database guarded by the software quality improvement group. It is easily visible to all members of the project for interpretation.

Software project telemetry exhibits graceful degradation. While complete telemetry data provides the best support for project management, the analyses should not be brittle: they should still provide value even if sensor data occasionally "drops out" during the project. Telemetry collection and analysis should provide decision-making value even if these activities start midway through a project.

Software project telemetry is used for in-process monitoring, control, and short-term prediction. Telemetry analyses provide representations of current project state and how it is changing at the time scales of days, weeks, or months. The simultaneous display of multiple project state values and how they change over the same time periods allow opportunistic analyses—the emergent knowledge that one state variable appears to co-vary with another in the context of the current project.

Software Project Telemetry enables a more incremental, distributed, visible, and experiential approach to project decision-making. It also creates perspectives on system development that can provide new insight into HPC development processes, as we illustrate in the case study below.

4. Process and Product measures for HPC utilizing Hackystat

The development of an HPC system from a software engineering perspective raises many interesting questions. How long does such a system take to develop? Do some components take longer to develop than others? How much of the system is devoted to the sequential code, and how much is devoted to the parallelization of this code? How did the developer allocate their time during development to these activities? Do different choices of HPC tools and technologies lead to different answers to these questions? Would a different application area lead to similar or different results?

We believe that automated infrastructure for the collection and analysis of product and process data is an important first step toward enabling the HPC community to generate answers to these questions, and then use these answers to improve the tools and techniques for HPC development. The question is, what process and product measures can be both automatically collected and used to provide interesting insight into the questions raise above? This case study investigates the use of the following measures: Active Time, Most Active File, Command Line Invocations, Parallel and Serial Lines of Code, Milestone Test Success, and Performance.

The following sections describe each of these measures and illustrate them with sample data from a one week "snapshot" of development of the Optimal Truss Design problem in our case study.

4.1. The Optimal Truss Design problem

Our case study focuses on the development of a system for optimal truss design. Specifically, the system finds a pin-connected steel truss structure that uses as little mass as possible to support a load connected from three attachment points on a wall to the load-bearing point away from the wall. This problem was originally developed for use in research on Purpose-Based Benchmarks (PBBs) [3]. PBBs gather a different and complementary set of metrics in order to assess productivity in terms of acceptability to the customer.

The system is being implemented by one of the authors, Michael Paulding, and thus generally conforms to the "lone researcher" workflow for HPC development. The system development process and associated case study started in the Spring of 2004 and is still ongoing. To date, the implementation of the Optimal Truss Design problem consists of approximately 1,200 source lines of code.

The solution to the Optimal Truss Design problem developed in this case study involves several components. The first component is termed the "sequential workhorse", which includes the task of solving a truss once all of its elements are defined. Solving a truss includes the calculation of its mass, which is determined by summing the mass of each of its components (e.g. all steel joints and members). In addition to mass calculation, solving a truss also includes verifying equilibrium and deformational constraints. Equilibrium constraints require that all forces and moments within a truss net zero magnitude, thus ensuring that the truss is not accelerating. Deformational constraints require that the length of members (strut or cable) used in the truss do not exceed construction safety limits. These limits are defined and known prior to runtime.

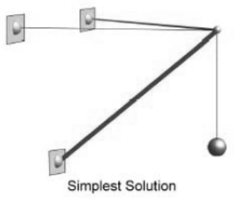


Figure 1. An unoptimized solution to the Truss problem

The second component of the Optimal Truss system generates the permutation of all possible truss topologies within the domain space, ensuring that the configuration with minimal mass is a global minimum. The domain space for the initial implementation is a 2-dimensional mesh of points, defining the rectangle formed between the attachment points and the load bearing point. Exploring all possible configurations results in a combinatorial explosion as the mesh size is increased and this served as the first point of parallelism in the implementation. The task of parallelizing topology generation can be equally divided among the available nodes. This can be accomplished in an "embarrassingly parallel" manner, where each row of the mesh is assigned to a different processor to permute.

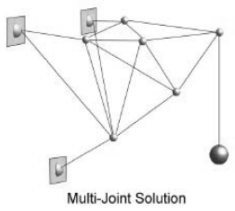


Figure 2. An optimized solution to the Truss problem

The third component of the system performs geometry assignment for all trusses. After generation, a topology defines the path of each truss from the attachment points to the load bearing point, but it does not specify what type of member connects each joint. In this stage, either a strut or a cable is substituted for each member, flushing out all permutations. Once the geometry has been assigned, it can be given to a processor to compute the mass of the truss and to determine whether the topology is valid under the equilibrium and deformational constraints.

Now that the description of the Optimal Truss Design problem, used in our case study, has been explained, it is prudent to illustrate and investigate the measures applied to the problem.

4.2. Active Time

Active Time is a measure of the time spent by developers editing source code (or other files) related to the system. Active Time can be collected automatically through the use of sensors attached to the editor used by developers. The sensors collect active time via a timer-based process inside the editor that wakes up every 30 seconds and checks to see if the active buffer has changed in identity or size since the last 30 seconds. If so, a timestamped "statechange" event is sent to the Hackystat server. Active Time does not reflect effort spent by developers on the project that does not involve editing files, including time spent viewing a file without performing editing actions. Support for non-editing activities such as “reading” is a subject of future research, but even the restricted definition of Active Time appears useful in the HPC context as a proxy for overall effort. For example, it helps a development team answer questions such as: *“How much of the overall development effort was spent editing files?”* or *“Did all team members devote equal time to writing code?”* or *“When was team effort focussed on code development during the project?”*

Figure 3 shows the Active Time associated with development of the Optimal Truss Design application for a sample period in May, 2004.

4.3. Most Active File

A measure related to Active Time is the “Most Active File”. One way to abstract the raw event stream sent from an editor-based Hackystat server begins by representing each day as a sequence of 288 five minute intervals. If a developer actively edits one or more files within a five minute period, then determine which file was edited most during that five minutes, and assign the “credit” for that five minute interval to that file and that file alone, which we call the “Most Active File”. (We performed a calibration study which found this to be a reasonable abstraction.) The Most Active File abstraction may be useful in the HPC context as a way of determining what specific files were the focus of developer attention, and how that focus of attention changed over the course of development.

For example, Figure 4 shows the Most Active Files associated with Optimal Truss Design during the first few days of this time interval.

4.4. Command Line Invocations

In addition to time spent editing files in an editor, HPC development frequently involves extensive use of shell processes to invoke programs such as make, gcc, etc. We have implemented a sensor for the Unix command shell (based upon the ’history’ shell mechanism) to record these command line invocations. Command Line Invocation data can be useful in the HPC context as a way of providing further insight into the types of activities performed by developers during the development of the HPC code. For example, if the HPC developer spends significant time working at the command line without concurrent editing of code, then it might be useful to develop an enhanced representation of Active Time that accounts for this type of effort as well. While the current sensor only captures command invocations and not their results, it might be useful to extend the sensor to capture the results of command line invocations in certain circumstances. For example, recording whether or not a compilation succeeded or failed as well as what types of run-time errors occur could help identify potential development bottlenecks.

Figure 5 illustrates Command Line Invocation data for a portion of one day during the development of the Optimal Truss Design system.

4.5. Parallel and Serial Size

To understand HPC software development, it helps to be able to represent both “serial” and “parallel” code. We have enhanced our size measurement tool, LOCC, with a token-based counter for C++ that allows us to count non-comment source lines of code, and determine for each line of code whether or not an MPI directive occurs on it. Thus, for HPC programs built using C++ and MPI, we can determine (a) the total number of files in the system, (b) the total non-commented size of each file in the system; (c) whether or not a file consists purely of serial (non-MPI) code or not; (d) for files containing MPI directives, the frequency of occurrence of each MPI directive; and (e) for files containing MPI code, what percentage of the non-comment source lines of code contained an MPI directive.

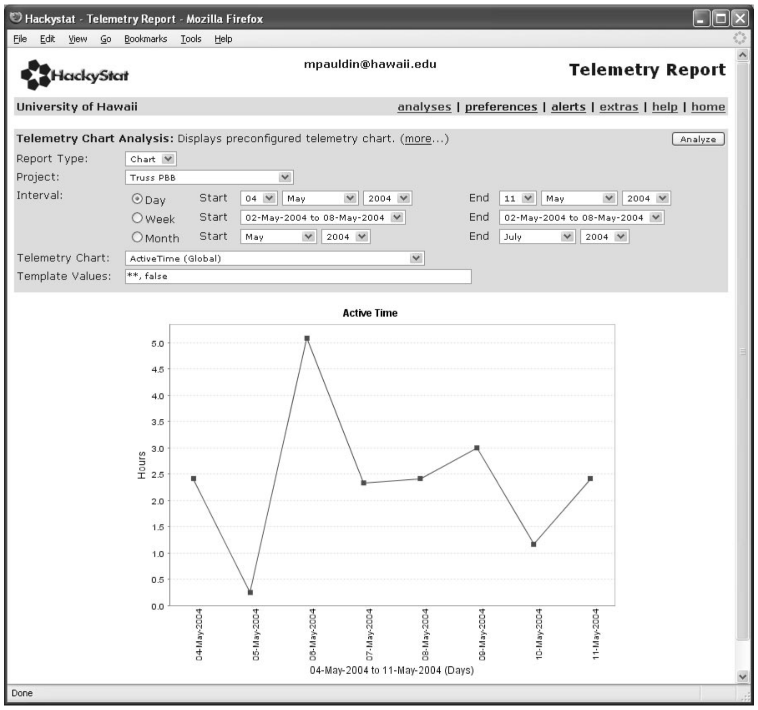
Figures 6, 7, and 8 provide perspectives on size data for the Optimal Truss Design system.

4.6. Functionality

We have defined a process for measuring the functionality of an HPC application and tracking its development progress through the use of unit testing. We have termed this approach “Progress Assessment through Milestone Tests” (PAMT).

Essentially, PAMT is a process in which HPC application designers draft the specification for the system as a set of unit tests prior to development. Each unit test is defined such that it represents a milestone, or significant aspect of application functionality. Quantitative interpretation of “significant” is determined by the application designer or program manager and is expected to vary between HPC projects. Defining the set of milestone tests prior to development provides a specification for the system and also serves as a mechanism to promote test driven design.

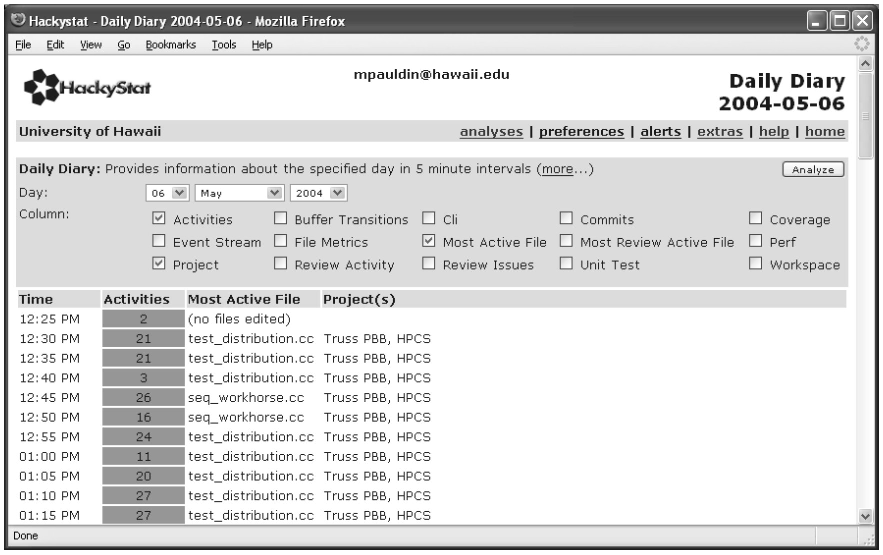
Once the milestone tests have been defined, the development team has a concrete set of tests to implement that, together, represent the functionality of the entire system. The development team can then implement the milestone tests in any order and their progress through the application can be monitored. System progress and functionality is measured by investigating the number


Hackystat - Telemetry Report - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
mpauldin@hawaii.edu
Telemetry Report
University of Hawaii
analyses | preferences | alerts | extras | help | home
Analyze
Active Time
Hours
04-May-2004
05-May-2004
06-May-2004
07-May-2004
08-May-2004
09-May-2004
10-May-2004
11-May-2004
04-May-2004 to 11-May-2004 (Days)
Done

Telemetry Chart Analysis: Displays preconfigured telemetry chart. (more...)

Report Type:	Chart				
Project:	Truss PBB				
Interval:	[✓ Day]	Start	04 May 2004	End	11 May 2004
	Week	Start	02-May-2004 to 08-May-2004	End	02-May-2004 to 08-May-2004
	Month	Start	May 2004	End	July 2004
Telemetry Chart:	ActiveTime (Global)				
Template Values:	**, false				

Figure 3. Active time


Hackystat - Daily Diary 2004-05-06 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
mpauldin@hawaii.edu
Daily Diary
2004-05-06
University of Hawaii
analyses | preferences | alerts | extras | help | home
Analyze
Done

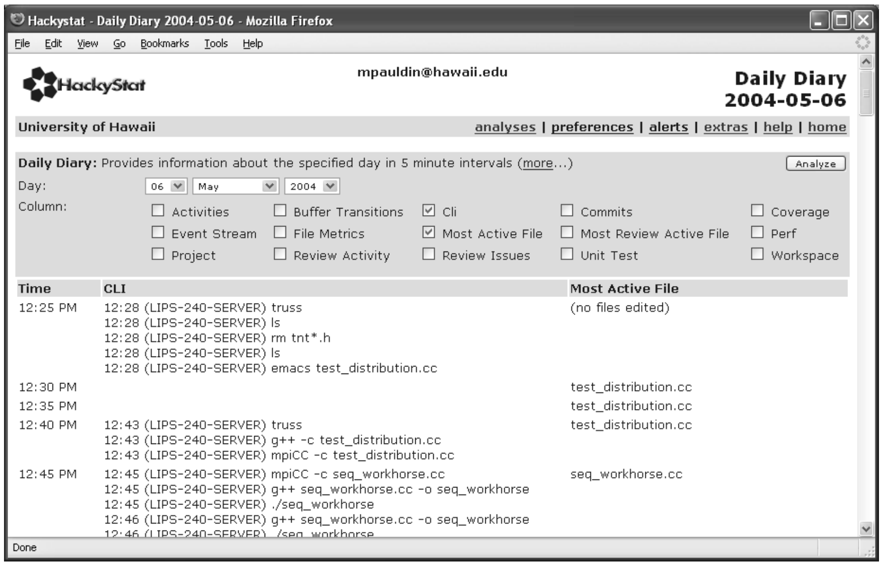
Daily Diary: Provides information about the specified day in 5 minute intervals (more...)

Day: 06 May 2004

Column: [✓ Activities] Buffer Transitions, Cli, Commits, Coverage, Event Stream, File Metrics, [✓ Most Active File] Most Review Active File, Perf, [✓ Project] Review Activity, Review Issues, Unit Test, Workspace

Time	Activities	Most Active File	Project(s)
12:25 PM	2	(no files edited)	
12:30 PM	21	test_distribution.cc	Truss PBB, HPCS
12:35 PM	21	test_distribution.cc	Truss PBB, HPCS
12:40 PM	3	test_distribution.cc	Truss PBB, HPCS
12:45 PM	26	seq_workhorse.cc	Truss PBB, HPCS
12:50 PM	16	seq_workhorse.cc	Truss PBB, HPCS
12:55 PM	24	test_distribution.cc	Truss PBB, HPCS
01:00 PM	11	test_distribution.cc	Truss PBB, HPCS
01:05 PM	20	test_distribution.cc	Truss PBB, HPCS
01:10 PM	27	test_distribution.cc	Truss PBB, HPCS
01:15 PM	27	test_distribution.cc	Truss PBB, HPCS

Figure 4. Most Active File


Hackystat - Daily Diary 2004-05-06 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
HackyStat
analyses | preferences | alerts | extras | help | home
Analyze
Done

mpauldin@hawaii.edu

Daily Diary 2004-05-06

University of Hawaii

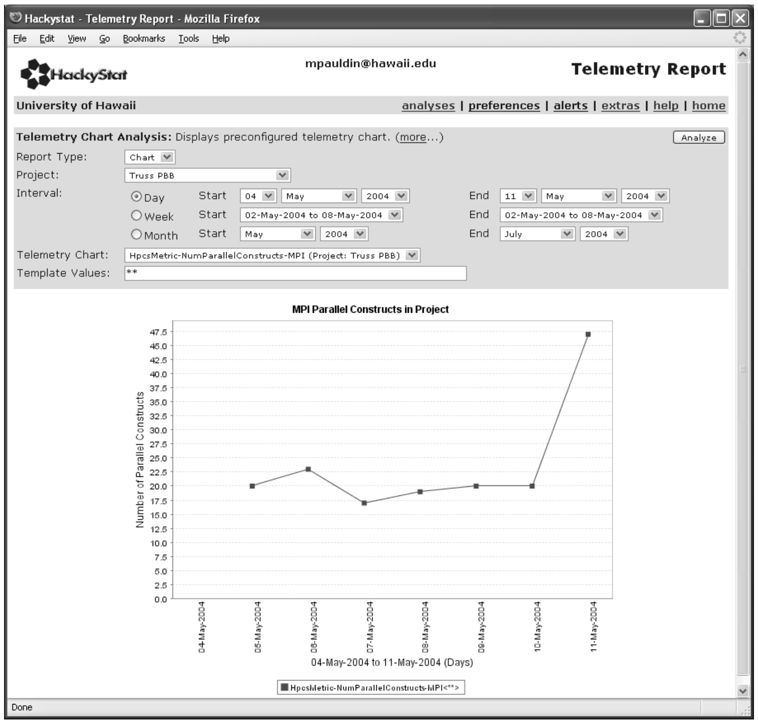
Daily Diary: Provides information about the specified day in 5 minute intervals (more...)

Day: 06 May 2004

Column: ☐ Activities ☐ Buffer Transitions ☑ Cli ☐ Commits ☐ Coverage ☐ Event Stream ☐ File Metrics ☑ Most Active File ☐ Most Review Active File ☐ Perf ☐ Project ☐ Review Activity ☐ Review Issues ☐ Unit Test ☐ Workspace

Time	CLI	Most Active File
12:25 PM	12:28 (LIPS-240-SERVER) truss 12:28 (LIPS-240-SERVER) ls 12:28 (LIPS-240-SERVER) rm tnt*.h 12:28 (LIPS-240-SERVER) ls 12:28 (LIPS-240-SERVER) emacs test_distribution.cc	(no files edited)
12:30 PM		test_distribution.cc
12:35 PM		test_distribution.cc
12:40 PM	12:43 (LIPS-240-SERVER) truss 12:43 (LIPS-240-SERVER) g++ -c test_distribution.cc 12:43 (LIPS-240-SERVER) mpiCC -c test_distribution.cc	test_distribution.cc
12:45 PM	12:45 (LIPS-240-SERVER) mpiCC -c seq_workhorse.cc 12:45 (LIPS-240-SERVER) g++ seq_workhorse.cc -o seq_workhorse 12:45 (LIPS-240-SERVER) ./seq_workhorse 12:46 (LIPS-240-SERVER) g++ seq_workhorse.cc -o seq_workhorse 12:46 (LIPS-240-SERVER) ./seq_workhorse	seq_workhorse.cc

Figure 5. Command Line Invocations


Hackystat - Telemetry Report - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
HackyStat
analyses | preferences | alerts | extras | help | home
Analyze
MPI Parallel Constructs in Project
Number of Parallel Constructs
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
04-May-2004
05-May-2004
06-May-2004
07-May-2004
08-May-2004
09-May-2004
10-May-2004
11-May-2004
04-May-2004 to 11-May-2004 (Days)
HpcsMetric-NumParallelConstructs-MPI<**>
Done

mpauldin@hawaii.edu

Telemetry Report

University of Hawaii

Telemetry Chart Analysis: Displays preconfigured telemetry chart. (more...)

Report Type: Chart

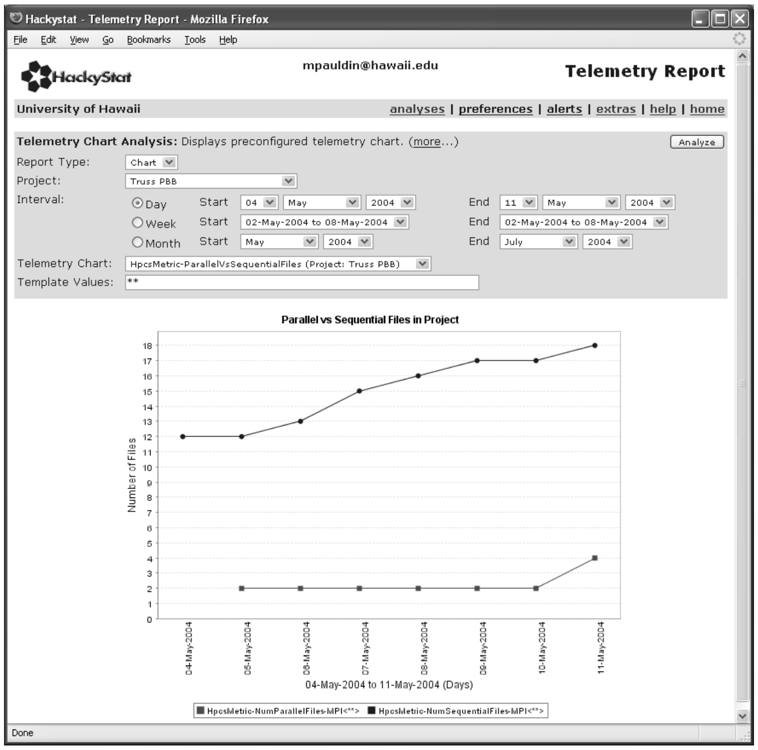
Project: Truss PBB

Interval: [✓ Day] Start 04 May 2004 End 11 May 2004
Week Start 02-May-2004 to 08-May-2004 End 02-May-2004 to 08-May-2004
Month Start May 2004 End July 2004

Telemetry Chart: HpcsMetric-NumParallelConstructs-MPI (Project: Truss PBB)

Template Values: **

Figure 6. Parallel vs. sequential constructs


Hackystat - Telemetry Report - Mozilla Firefox

mpauldin@hawaii.edu

Telemetry Report

University of Hawaii

Telemetry Chart Analysis: Displays preconfigured telemetry chart. (more...)

Report Type: Chart

Project: Truss PBB

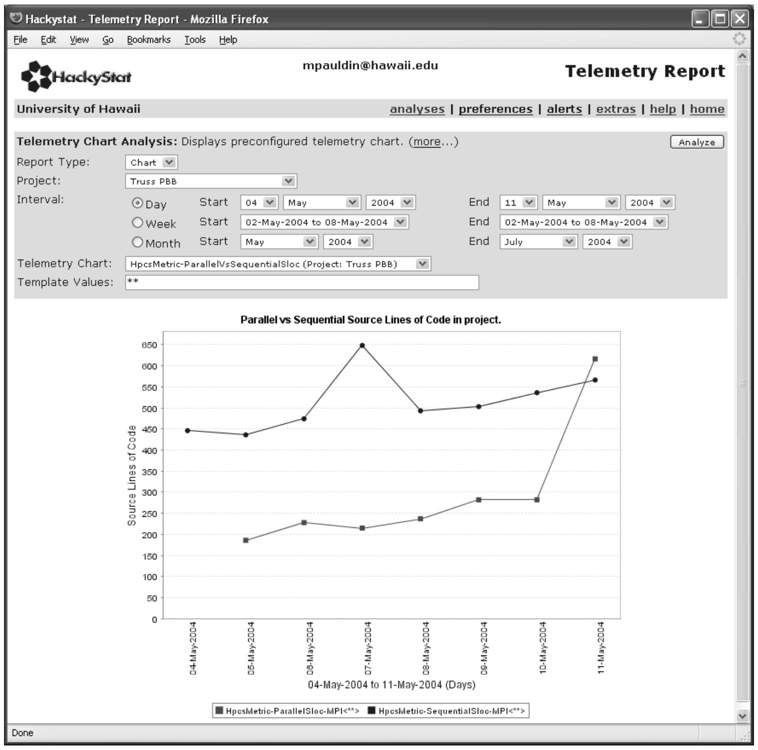
Interval: (● Day) Start 04 May 2004 End 11 May 2004

Telemetry Chart: HpcsMetric-ParallelVsSequentialFiles (Project: Truss PBB)

Template Values: **

File
Edit
View
Go
Bookmarks
Tools
Help
analyses | preferences | alerts | extras | help | home
Analyze
Week
Month
Start
02-May-2004 to 08-May-2004
End
02-May-2004 to 08-May-2004
May
2004
July
Parallel vs Sequential Files in Project
Number of Files
04-May-2004
05-May-2004
06-May-2004
07-May-2004
08-May-2004
09-May-2004
10-May-2004
11-May-2004
04-May-2004 to 11-May-2004 (Days)
HpcsMetric-NumParallelFiles-MPI<**>
HpcsMetric-NumSequentialFiles-MPI<**>
Done

Figure 7. Parallel vs. Sequential Files


Hackystat - Telemetry Report - Mozilla Firefox

mpauldin@hawaii.edu

Telemetry Report

University of Hawaii

Telemetry Chart Analysis: Displays preconfigured telemetry chart. (more...)

Report Type: Chart

Project: Truss PBB

Interval: (● Day) Start 04 May 2004 End 11 May 2004

Telemetry Chart: HpcsMetric-ParallelVsSequentialSloc (Project: Truss PBB)

Template Values: **

File
Edit
View
Go
Bookmarks
Tools
Help
analyses | preferences | alerts | extras | help | home
Analyze
Week
Month
Start
02-May-2004 to 08-May-2004
End
02-May-2004 to 08-May-2004
May
2004
July
Parallel vs Sequential Source Lines of Code in project.
Source Lines of Code
04-May-2004
05-May-2004
06-May-2004
07-May-2004
08-May-2004
09-May-2004
10-May-2004
11-May-2004
04-May-2004 to 11-May-2004 (Days)
HpcsMetric-ParallelSloc-MPI<**>
HpcsMetric-SequentialSloc-MPI<**>
Done

Figure 8. Parallel vs. Sequential SLOC

of milestone tests passing in ratio to the total number of milestone tests representing the system. In most cases, a development team will begin implementation with zero milestone tests passing and finish development when all milestone tests pass.

For the Optimal Truss Design problem, a set of 10 milestone tests were defined prior to implementation. Individually, each test represents a significant functionality of the application and together they provide a specification for the entire system. For the Optimal Truss problem, the milestone tests were written in CppUnit, a unit test framework for the C++ programming language. Below is an example of a single milestone test for the Optimal Truss problem.

> **Milestone Test 4:** This test verifies that the application is capable of representing a 2-dimensional topology. In the Optimal Truss specification, a topology is defined as a set of 2 trusses that individually connect the 2 attachment points to the load bearing point. Interconnections (members) between the trusses are allowed. Therefore, for this milestone test, given 2 attachment points, a load bearing point and the number of joints, the application must be able to query:
>
> 1. Each of the trusses connecting the 2 attachment points to the load bearing point
> 2. The set of members composing one of the trusses in the topology
> 3. Given a truss, whether it is part of the topology

From this chart is is evident that during the development period from 04-May-2004 through 11-May-2004 that the Optimal Truss application progressed from 1 milestone test passing at the beginning of the interval to 5 milestone tests passing at the end. It is important to note that this interval represents a sample of the development period and does not capture start to finish. In addition, this trend indicates a consistent increase in passing milestone tests. However, it is quite possible for development to lower the number of successful milestone tests, indicated by a negative slope in the trend.

4.7. Performance

The high performance computing community has developed a broad range of standard measures to characterize parallel performance, including degree of parallelism, average parallelism, speedup, redundancy, and utilization. In this research, we are not attempting to specify the “right” performance measure for any particular application area. Instead, we advocate that performance be measured regularly throughout development using as many metrics as necessary to best characterize the application.

Performance measures are not generally interesting as absolute numbers, since the absolute values are obviously dependent upon current hardware and other physical resources. Performance measures are interesting as relative numbers, in the sense that the way they change over time tells us whether or not and to what extent developers could tune an initial implementation to improve its performance, how the code evolved to obtain this performance increase, and whether or not functionality was sacrificed in order to do so.

Figure 10 shows the execution (wall time) performance of the Optimal Truss Design system developed in the case study for a sample time interval.

5. Conclusions

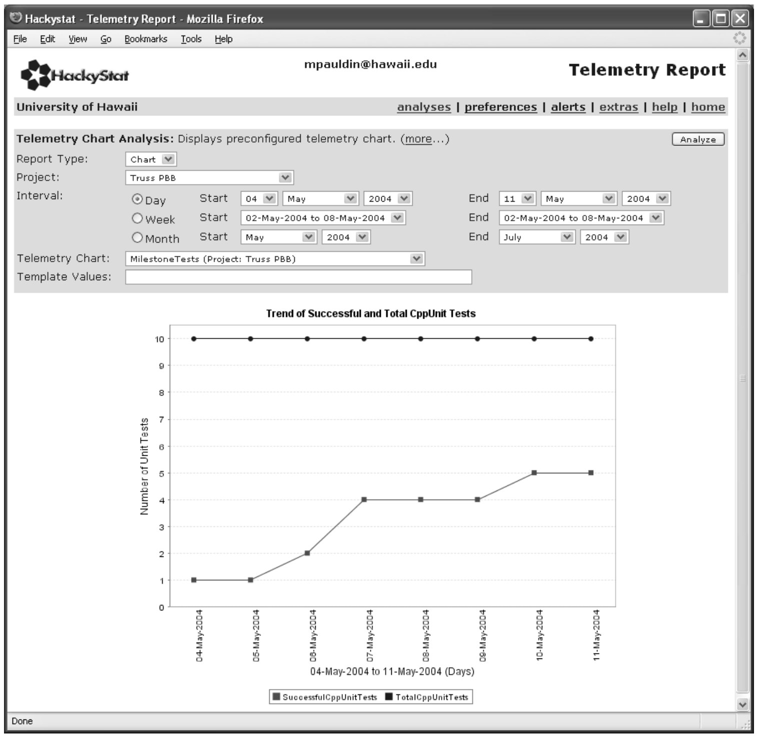
After accumulating the data trends provided by the Hackystat system, we are able to gain insights that assist in understanding the development of HPC applications. From the graphs presented earlier, we are able to make interpretations about development activities, development progress and application performance and functionality tradeoffs.

5.1. Development Activities

From the data presented in Figure 5 we are able to interpret the developer activities of that particular day. The daily dairy for 06-May-2004 lists all the commands issued to the console on that day. Figure 5 is a sample of all the command line invocations issued, but it provides insight to the developer activities on that day.

For example, from the command line invocations and most active file data for 06-May-2004, we can observe that the developer devoted the most time on the *test_distribution.cc* file. It so happens that this file implements the distribution of 2-dimensional mesh from which the truss topologies are constructed. Furthermore, the distribution of topologies is a significant function of the Optimal Truss problem and has been designated as a milestone test of the system. Therefore, from this data, an observer can conclude that the developer was investing his efforts on implementing functionality on this day, rather than on increasing performance by optimizing code.

In addition, an observer, such as a project manager or the developer himself, can observe the active time trend in Figure 3 to understand the time invested to implement


Hackystat - Telemetry Report - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
HackyStat
analyses | preferences | alerts | extras | help | home
Analyze
Trend of Successful and Total CppUnit Tests
Number of Unit Tests
04-May-2004 to 11-May-2004 (Days)
SuccessfulCppUnitTests
TotalCppUnitTests
Done

mpauldin@hawaii.edu

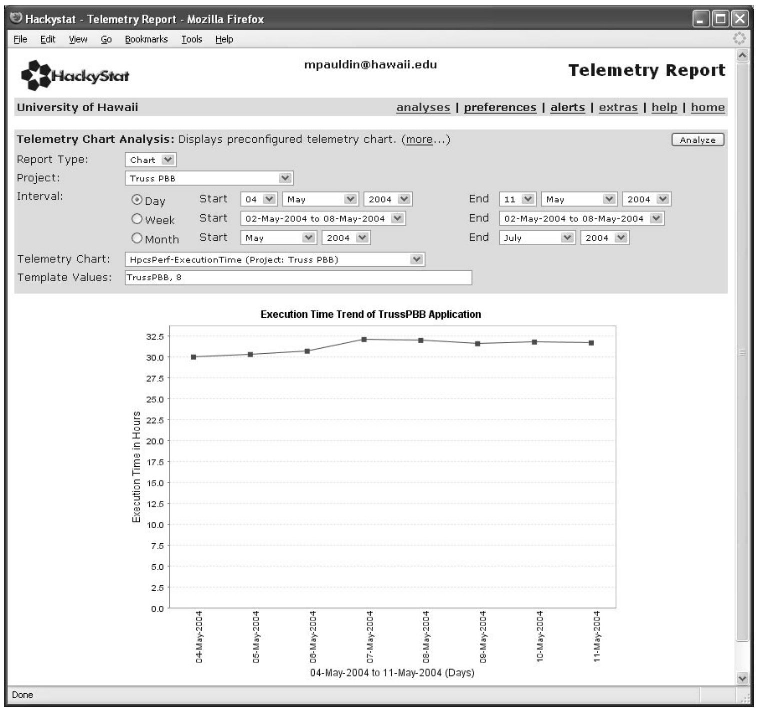
Telemetry Report

University of Hawaii

Telemetry Chart Analysis: Displays preconfigured telemetry chart. (more...)

Report Type:	Chart			
Project:	Truss PBB			
Interval:	(•) Day	Start 04 May 2004	End 11 May 2004	
	( ) Week	Start 02-May-2004 to 08-May-2004	End 02-May-2004 to 08-May-2004	
	( ) Month	Start May 2004	End July 2004	
Telemetry Chart:	MilestoneTests (Project: Truss PBB)			
Template Values:				

Figure 9. Progress Assessment through Milestone Tests


Hackystat - Telemetry Report - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
HackyStat
analyses | preferences | alerts | extras | help | home
Analyze
Execution Time Trend of TrussPBB Application
Execution Time in Hours
04-May-2004 to 11-May-2004 (Days)
Done

mpauldin@hawaii.edu

Telemetry Report

University of Hawaii

Telemetry Chart Analysis: Displays preconfigured telemetry chart. (more...)

Report Type:	Chart		
Project:	Truss PBB		
Interval:	(•) Day	Start 04 May 2004	End 11 May 2004
	( ) Week	Start 02-May-2004 to 08-May-2004	End 02-May-2004 to 08-May-2004
	( ) Month	Start May 2004	End July 2004
Telemetry Chart:	HpcsPerf-ExecutionTime (Project: Truss PBB)		
Template Values:	TrussPBB, 8		

Figure 10. Truss Execution Time Performance

a particular milestone. For example, in Figure 3 on 06-May-2004, it is evident that the developer spent over 5 hours editing code to implement the topology distribution milestone.

5.2. Development Progress

The data presented in Progress Assessment through Milestone Tests (PAMT) chart, as illustrated in Figure 9, provides a clear illustration of the real-time progress being made on the Optimal Truss problem.

There are two trends presented in this figure, one representing the total number of milestone tests defined for the Optimal Truss problem and the other representing the number of milestone tests passing at the conclusion of each day.

In the Optimal Truss problem, the total milestone tests are represented by the horizontal line fixed at 10 unit tests. This indicates that there are 10 milestone tests encompassing the Optimal Truss problem and that the project manager has not added or removed any of these milestones during this time interval. It is quite possible that a project manager may have to alter milestones in order to meet deadlines and this analysis provides a trend for this purpose. For example, if the total milestone tests is altered, the total tests trend on the chart will move up or down accordingly.

The PAMT chart also allows an observer to track the progress made through the application. For example, in this figure, the lower trend represents the number of milestone tests passing on each day. Every time a new milestone test passes, it indicates that another unit of functionality has been added to the system provided that all previously passing tests still pass after the change.

Coupling the PAMT data with the active time data in Figure 3, an observer is also able to interpret a quantitative measurement of how much development time was devoted to a particular unit of functionality. For example, on 06-May-2004, approximately 5 hours of editing were invested to add one unit of functionality to the application. This is indicated by the number of passing milestone tests increasing from one to two on this day. In addition, an observer can quickly understand the percentage complete of the system. On 06-May-2004, the Optimal Truss problem has 2 out of 10 milestone tests passing and is therefore 20% complete.

5.3. Application Performance and Functionality Tradeoffs

When one combines the data presented in the Performance chart in Figure 10 with the PAMT Functionality chart in Figure 9, it reveals an example of the interactions between performance and functionality in HPC development.

One of the primary objectives of HPC development is to obtain the fastest possible execution time of the system. This goal influences developers to frequently (if not constantly) think about or perform optimization on their code.

However, as functionality is added to the application, it is common for the performance of the system to decrease, indicated by an increase in execution time.

The data presented in figures 10 and 9 reveal this performance and functionality tradeoff. For example, execution time between 04-May-2004 and 07-May-2004 increaseses roughly from 30.0 hours to 32.5 hours. On the other hand, 3 additional milestone tests, representing system functionality, are implemented successfully during this interval. This indicates that three units of functionality have been added at the cost of an addition of approximately 10% in execution time.

Data presented in these figures allow project managers to understand how functionality increases affect system performance. It also gives them a starting point to determine which functionality should be optimized in the case where the performance degradation is not acceptable. Trends such as these enable project managers and developers to understand the development process and make in-process decisions to affect the development outcome.

In conclusion, we have found that Active Time, Most Active File, Command Line Invocations, Parallel and Serial Lines of Code, Milestone Test Success, and Performance constitute an interesting set of process and product measures that can be automatically collected during HPC development. As our case study continues, we will look for other opportunities to use this measures to gain insight into opportunities to improve high performance computing.

6. Acknowledgements

We would like to thank the University of Hawaii and MHPCC (Maui High Performance Computing Center) for their continued support of and interest in our research.

References

[1] The DARPA high productivity computing systems program. http://www.highproductivity.org/.

[2] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[3] J. Gustafson. Purpose Based Benchmarks. *International Journal of High Performance Computing Applications*, 12(1):14, 2004.

[4] W. S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, New York, 1995.

[5] P. M. Johnson and A. M. Disney. The personal software process: A cautionary case study. *IEEE Software*, 15(6), November 1998.

[6] P. M. Johnson, H. Kou, J. M. Agustin, C. Chan, C. A. Moore, J. Miglani, S. Zhen, and W. E. Doane. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the 2003 International Conference on Software Engineering*, Portland, Oregon, May 2003.

[7] P. M. Johnson, C. A. Moore, J. A. Dane, and R. S. Brewer. Empirically guided software effort guesstimation. *IEEE Software*, 17(6), December 2000.

[8] D. A. Reed, editor. *The Roadmap for the Revitalization of High-End Computing*. Computing Research Association, 2003.