Software Development Environments for Scientific and Engineering Software: A Series of Case Studies

Jeffrey C. Carver
University of Alabama

Los Alamos Computer Science Symposium
October 15, 2008

Outline

- Introduction
  - Software Engineering
  - HPCS project
- Methodology
  - Process
  - Projects Studied
- Results
  - Lessons Learned
  - Summary

Introduction

- Software engineering is an engineering discipline
- We need to understand products, processes, and the relationship between them (we assume there is one)
- We need to conduct human-based studies (case studies and experiments)
- We need to package (model) that knowledge for use and evolution
- Recognizing these needs changes how we think, what we do, what is important, and the nature of the discipline

Empirical Studies

- Understanding a Discipline
  - Building Models
    - application domain, workflows, problem solving processes
  - Checking Understanding
    - testing models, experimenting in the real world
  - Analyzing Results
    - learn, encapsulate knowledge and refine models
  - Evolving Models

- The empirical paradigm has been used in many other fields, e.g. physics, medicine, manufacturing

High Productivity Computing Systems (HPCS)

- Problem: How do you build sufficient knowledge about high end computing (HEC) so you can improve the time and cost of developing these codes?

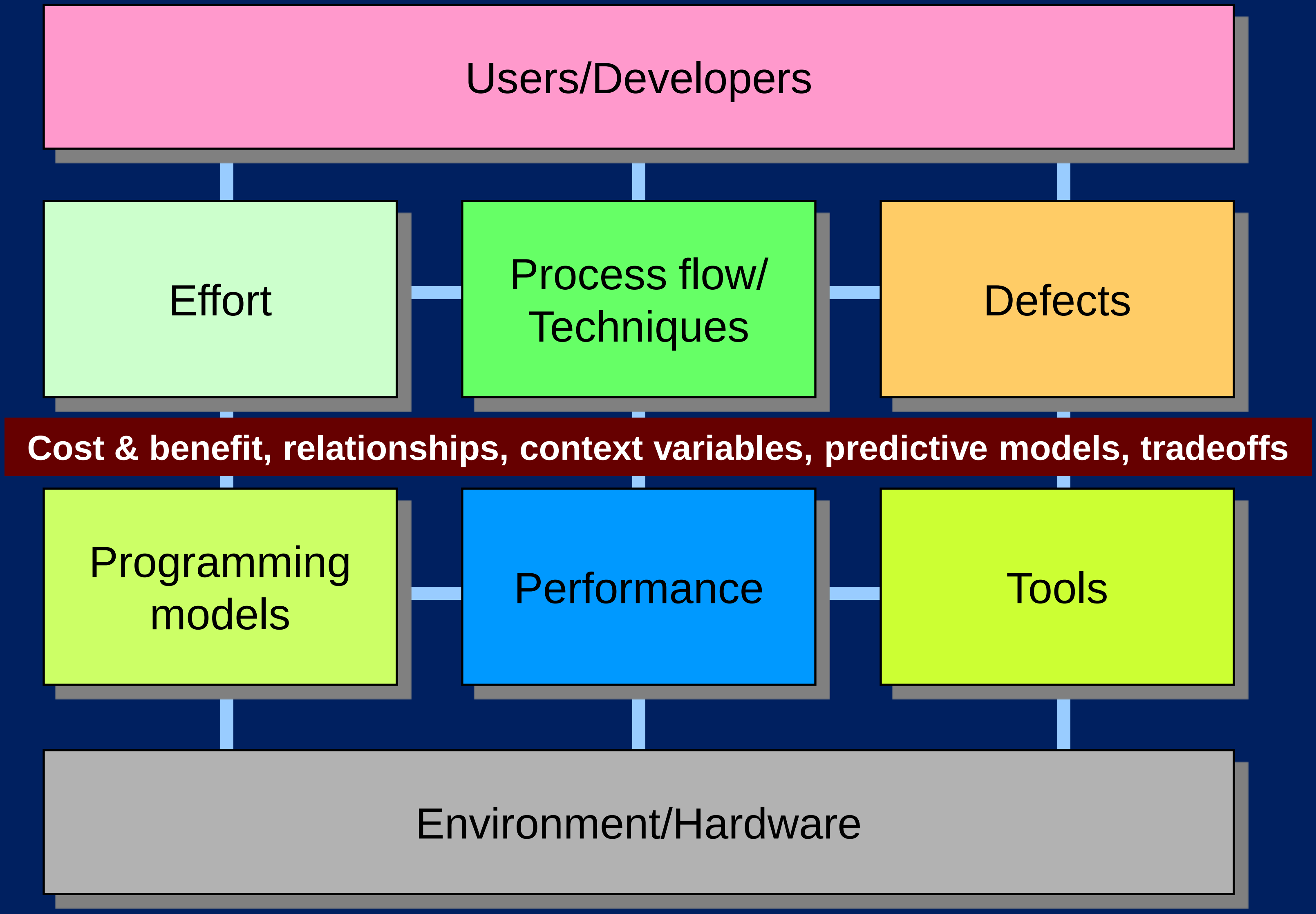
- Project Goal: Improve the buyer's ability to select the high end computer for the problems to be solved based upon productivity, where productivity means

  Time to Solution = Development Time + Execution Time

- Research Goal: Develop theories, hypotheses, and guidelines that allow us to characterize, evaluate, predict and improve how an HPC environment (hardware, software, human) affects the development of high end computing codes.

- Partners: MIT Lincoln Labs, MIT, MSU, UCSD, UCSB, UCSD, UH, UMD, UNL, USC, FC-MD, ISU

Areas of Study



Types of HPCS Studies

Controlled experiments

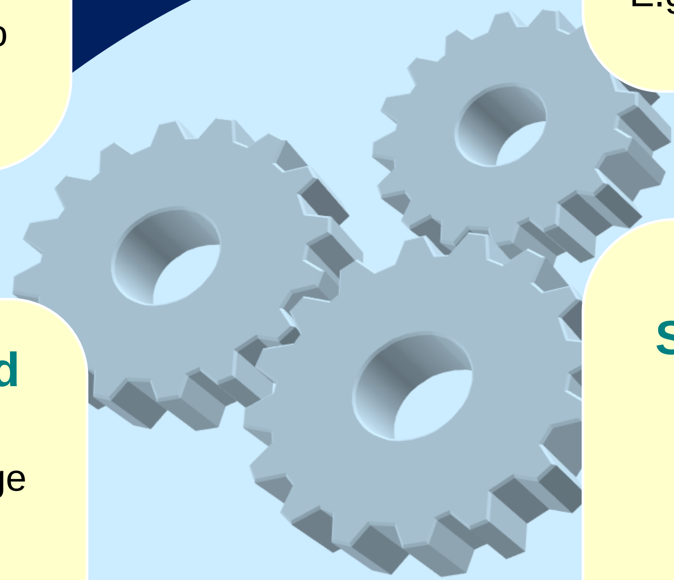
Study programming in the small under controlled conditions to:
Identify key variables, check out methods for data collection, get professors interested in empiricism

E.g., compare effort required to develop code in MPI vs. OpenMP

Observational studies

Characterize in detail a realistic programming problem in realistic conditions to:
validate data collection tools and processes

E.g., build an accurate effort data model



Case studies and field studies

Study programming in the large under typical conditions

E.g., understand multi-programmer development workflow

Surveys, interviews & focus groups

Collect “folklore” from practitioners in government, industry and academia

e.g., generate hypotheses to test in experiments and case studies

Current Study

- Environment
  - Computational Science and Engineering projects
- Goals
  - Understand and document software development practices
  - Gather initial information about what practices are effective / ineffective
- Approach
  - Series of retrospective case studies

Case Study Methodology

Projects Studied:
FALCON

GOAL: Develop a predictive capability for a product whose performance involves complex physics to reduce the dependence of the sponsor on expensive and dangerous tests.

DURATION: ~10 years

STAFFING: 15 FTEs

LANGUAGE: OO-FORTRAN

CODE SIZE: ~405 KSLOC

USERS: External; highly knowledgeable product engineers

TARGET PLATFORM:
- Shared-memory LINUX cluster (~2000 nodes)
- Vendor-specific shared-memory cluster (~1000 nodes)

Post, 2005

Projects Studied:
HAWK

GOAL: Develop a computationally predictive capability to analyze the manufacturing process allowing the sponsor to minimize the use of time-consuming expensive prototypes for ensuring efficient product fabrication.

DURATION: ~ 6 Years

STAFFING: 3 FTEs

USERS: Internal and external product engineers; small number

LANGUAGE: C++ (67%); C (18%); FORTRAN90/Python (15%)

CODE SIZE: ~134 KSLOC

TARGET PLATFORM:
- SGI (Origin 3900)
- Linux Networx (Evolocity Cluster)
- IBM (P-Series 690 SP)
- Intel-based Windows platforms

Kendall, 2005a

Projects Studied:
CONDOR

GOAL: Develop a simulation to analyze the behavior of a family of materials under extreme stress allowing the sponsor to minimize the use of time-consuming expensive and infeasible testing.

DURATION: ~ 20 Years

STAFFING: 3-5 FTEs

LANGUAGE: FORTRAN77 (85%)

CODE SIZE: ~200 KSLOC

USERS: Internal and external; several thousand occasional users; hundreds of routine users

TARGET PLATFORM:
- PC – running 106 cells for a few hours to a few days (average)
- Parallel application – 108 cells on 100 to a few 100s of processors

Kendall, 2005b

Projects Studied:
EAGLE

GOAL: Determine if parallel, real-time processing of sensor data is feasible on a specific piece of HPC hardware deployed in the field

DURATION: ~ 3 Years

LANGUAGE: C++

STAFFING: 3 FTEs

CODE SIZE: < 100 KSLOC

USERS: Demonstration project – no users

TARGET PLATFORM:
- Specialized computer that can be deployed on military platforms
- Developed on – SUN Sparcs (Solaris) and PC (Linux)

Kendall, 2006

Projects Studied:
NENE

GOAL: Calculate the properties of molecules using a variety of computational quantum mechanical models

DURATION: ~25 Years

LANGUAGE: FORTRAN77 subset of FORTRAN90

STAFFING: ~10 FTEs (Thousands of contributors)

CODE SIZE: 750 KSLOC

USERS: 200,000 installations and estimated 100,000 users

TARGET PLATFORM: All commonly used platforms except Windows-based PCs

Projects Studied:
OSPREY

GOAL: One component of a large weather forecasting suite that combines the interactions of large-scale atmospheric models with large-scale oceanographic models.

DURATION: ~10 years (predecessor > 25 years)

LANGUAGE: FORTRAN

STAFFING: ~10 FTEs

CODE SIZE: 150 KLOC (50 KLOC Comments)

USERS: Hundreds of installations – some have hundreds of users

TARGET PLATFORM: SGI, IBM, HP, and Linux

Kendall, 2008

Projects Studied:
Summary

	FALCON	HAWK	CONDOR	EAGLE	NENE	OSPREY
Application Domain	Product Performance	Manufacturing	Product Performance	Signal Processing	Process Modeling	Weather Forecasting
Duration	~ 10 years	~ 6 years	~ 20 years	~ 3 years	~ 25 years	~10 years
# of Releases	9 (production)	1	7	1	?	?
Staffing	15 FTEs	3 FTEs	3-5 FTEs	3 FTEs	~10 FTEs (100's of contributors)	~10 FTEs
Customers	< 50	10s	100s	None	~ 100,000	100s
Code Size	~ 405,000 LOC	~ 134,000 LOC	~200,000 LOC	< 100,000 LOC	750,000 LOC	150,000 LOC
Primary Languages	F77 (24%), C (12%)	C++ (67%), C (18%)	F77 (85%)	C++, Matlab	F77 (95%)	Fortran
Other Languages	F90, Python, Perl, ksh/csh/sh	Python, F90	F90, C, Slang	Java Libraries	C	C
Target Hardware	Parallel Supercomputer	Parallel Supercomputer	PCs to Parallel Supercomputer	Embedded Hardware	PCs to Parallel Supercomputer	Parallel Supercomputer

Lessons Learned

Lessons Learned: Validation and Verification

Validation

- Does the software correctly capture the laws of nature?
- Hard to establish the correct output of simulations *a priori*
  - Exploring new science
  - Inability to perform experimental replications

Verification

- Does the application accurately solve the equations of the solution algorithm?
- Difficult to identify problem source
  - Creation of mathematical model by domain expert
  - Translation of mathematical model into algorithm(s)
  - Implementation of algorithms in software

Lessons Learned:
Validation and Verification

I have tried to position CONDOR to the place where it is kind of like your trusty calculator – it is an easy tool to use. Unlike your calculator, it is only 90% accurate … you have to understand that then answer you are going to get is going to have a certain level of uncertainty in it. The neat thing about it is that it is easy to get an answer in the general sense <to a very difficult problem>.

- Implications
  - Traditional methods of testing software then comparing the output to expected results are not sufficient
  - These developers need additional methods to ensure quality and limits of software

Lessons Learned: Language Stability

- Long project lifecycles require code that is:
  - Portable
  - Maintainable
- FORTRAN
  - Easier for scientists to learn than C++
  - Produces code that performs well on large-scale supercomputers
- Users of the code interact frequently with the code
- Implications
  - FORTRAN will dominate for the near future
  - New languages have to have benefits of FORTRAN plus some additional benefits to be accepted

Lessons Learned:
Use of Higher-Level Languages

I'd rather be closer to machine language than more abstract. I know even when I give very simple instructions to the compiler, it doesn't necessarily give me machine code that corresponds to that set of instructions. If this happens with a simple do-loop in FORTRAN, what happens with a monster object-oriented thing?

- MATLAB
  - Code is not efficient or fast enough
  - Used for prototyping

- C++
  - Used by some newer teams
  - Mostly used the C subset of C++

- Implications
  - These developers place more constraints on the language that traditional IT developers
  - A language has to
    - Be easy to learn
    - Offer reasonably high performance
    - Exhibit stability
    - Give developers confidence in output of compiler

Lessons Learned: Development Environments

They all [the IDEs] try to impose a particular style of development on me and I am forced into a particular mode

- Developers prefer flexibility of the command line over an Integrated Development Environment (IDE). They believe that:
  - IDEs impose too much rigidity
  - They are more efficient when typing commands than when navigating menus

- Implications – developers do not adopt IDEs because:
  - They do not trust the IDE to automatically perform a task in the same way they would do it manually
  - They expect greater flexibility than is currently provided
  - Prefer to use what they know rather than change

Lessons Learned:
External Software

- Projects view external software as a risk
  - Long duration
  - Fear that software may disappear or become unsupported
  - Prefer to develop tools in-house or use open-source

- Exception – NENE
  - Employed a librarian to thoroughly test code before integrating into code base
  - Designed the project so that it was not dependent on external software to meet its commitments

- Implication - Tool problem
  - Very few quality tools for this environment
  - Catch-22 situation

Lessons Learned: Development Goals

- Multiple goals are important
  - Performance – software is used on supercomputer
  - Portability and Maintainability – platforms change multiple times during a project

- Success of a project depends on the ability to port software to new machines

- Implications
  - The motivation for these projects may be different than for traditional IT projects
  - Methods must be chosen and tailored to align with the overall project goals

Lessons Learned:
Agile vs. Traditional Methodologies

- "Agile" refers to the philosophical approach rather than to any particular Agile method

- Projects are often doing new science, so the requirements cannot be known upfront

- Teams have been operating with an agile philosophy before they even knew what it was – favoring individuals and good practices over rigid processes and tools

- Implications
  - Existing SE methodologies need to be tailored for this community
  - Rigid, process-heavy approaches are not used; both for technical and cultural reasons

Lessons Learned: Team Composition

In these types of high performance, scalable computing [applications], in addition to the physics and mathematics, *computer science plays a very major role. Especially when looking at optimization, memory management and making [the code] perform better … You need a multi-disciplinary team. It [C++] is not a trivial language to deal with … You need an equal mixture of subject theory, the actual physics, and technology expertise.*

- Complex problems and domains
  - Too difficult for most software engineers to understand quickly
  - Easier to teach domain scientists/engineers how to program
- Software engineers help with performance and flexibility
- Implication
  - Multi-disciplinary teams are important

Lessons Learned:
Key to Success

- Keeping customers (and sponsors) satisfied

- Lesson not unique to this community, but some constraints are important
  - Funding may come from one agency, while customers are members of another agency
  - Success depends on keeping both groups happy
  - HAWK project was suspended due to lack of customer support, even though it was a technical success for the funding agency

- Implication
  - Balancing the needs of these various stakeholders can be challenging

Summary

- Six case studies of computational science and engineering software
  - Projects sponsored by the US Federal Government and the National Science Foundation
  - Different domains and different goals

- Nine lessons learned about the programming environment drawn across all studies

- Contributions
  - For the software engineering community
    - Highlighted some reasons why the development process is different for this type of software
    - Provided insight into why traditional SE approaches are not used
  - For the computational science and engineering community
    - Provided ideas to guide the improvement of the software engineering process

Collaborators

- Doug Post - HPCMP
- Richard Kendall – SEI
- Dale Henderson – Los Alamos (retired)
- Andrew Mark – HPCMP
- David Fisher – HPCMP
- Clifford Rhoades, Jr. – Maui HPC Center
- Susan Squires – Tactics (formally with SUN)
- Christine Halverson - IBM

For More Information

- *IEEE Software* special issue – Developing Scientific Software (July/August 2008)
- Carver, et al. "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies." *ICSE 2007*
- ICSE workshops
  - Software Engineering for HPC Applications (2004-2005, 2007)
  - Software Engineering for Computational Science (2008)
    - Forthcoming news article in CiSE (March/April)
  - 2009 workshop proposed

Selected References

- Carver, J., Kendall, R., Squires, S. and Post, D. "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies." *Proceedings of the 29th International Conference on Software Engineering*. Minneapolis, USA. May 23-25, 2007. p. 550-559.
- Kendall, R., Carver, J., Fisher, D., Henderson, D., Mark, A., Post, D., Rhoades, C. and Squires, S. "Development of a Weather Forecasting Code: A Case Study." *IEEE Software*, July/August 2008. p. 59-65.
- Kendall, R., Post, D., Carver, J., and Squires, S. "Case Study of the Eagle Code Project." Los Alamos Technical Report, LA-UR-06-1092. 2006.
- Kendall, R., Carver, J., Mark, A., Post, D., Squires, S., and Shaffer, D. "Case Study of the Hawk Code Project." Los Alamos Technical Report, LA-UR-05-9011. 2005.
- Kendall, R.P., Mark, A., Post, D., Squires, S., and Halverson, C. Case Study of the Condor Code Project. Technical Report, LA-UR-05-9291. Los Alamos National Laboratories: 2005.
- Post, D.E., Kendall, R.P., and Whitney, E. "Case study of the Falcon Project". *Proceedings of Second International Workshop on Software Engineering for High Performance Computing Systems Applications (Held at ICSE 2005)*. St. Louis, USA. 2005. p. 22-26

Thank You!

Software Development Environments for Scientific and Engineering Software: A Series of Case Studies

Jeffrey Carver
University of Alabama
carver@cs.ua.edu