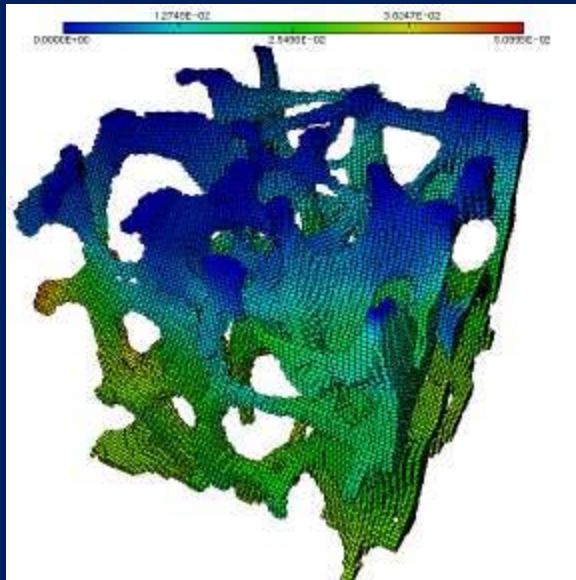


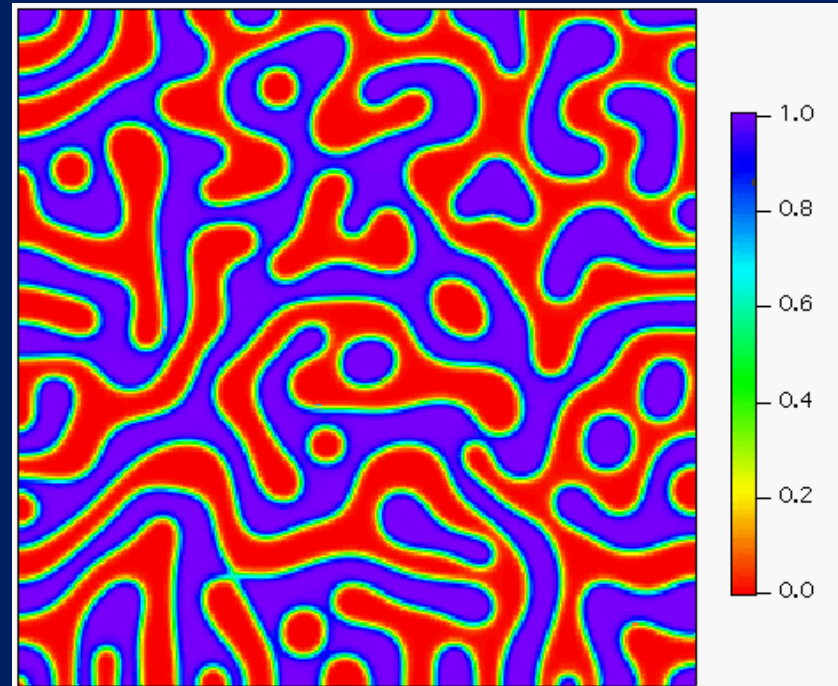
Software Development Environments for
Scientific and Engineering Software:
A Series of Case Studies

Jeffrey C. Carver
University of Alabama

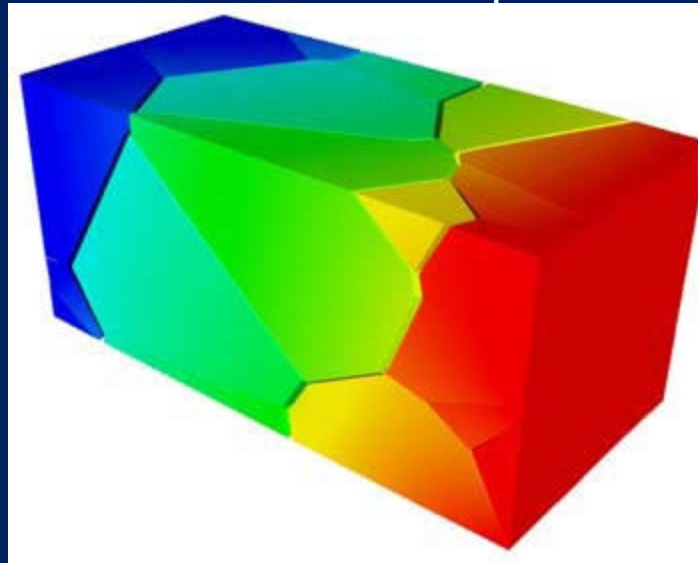
Trilinos User Group Meeting
November 4, 2009



<http://parfe.sourceforge.net/>



<http://www.ctcms.nist.gov/fipy/installation.html>



FlexFEM: http://www.cs.sandia.gov/materials_methods/news1.html

Outline

- Introduction
- Methodology
 - Empirical Software Engineering
 - Case Study
- Projects Studied
- Lessons Learned
- SE-CSE Workshops
- Summary

Introduction

- Scientific and engineering software
 - Simulations of physical phenomena
 - Processing large amounts of data
 - Performing complex calculations
- Unique characteristics
 - Requirements discovery and gathering process
 - Focus on science/engineering not software
 - Developers tend to be “process-averse”



Areas of Study



Effort

- How to measure?
- What variables affect?
- Relationship between effort and other variables?
- What activities consume effort?



Process Flow

- What is the normal process?
- Work vs. rework?
- Can automated data collection be used to measure process steps?
- Which techniques are effective / not effective?

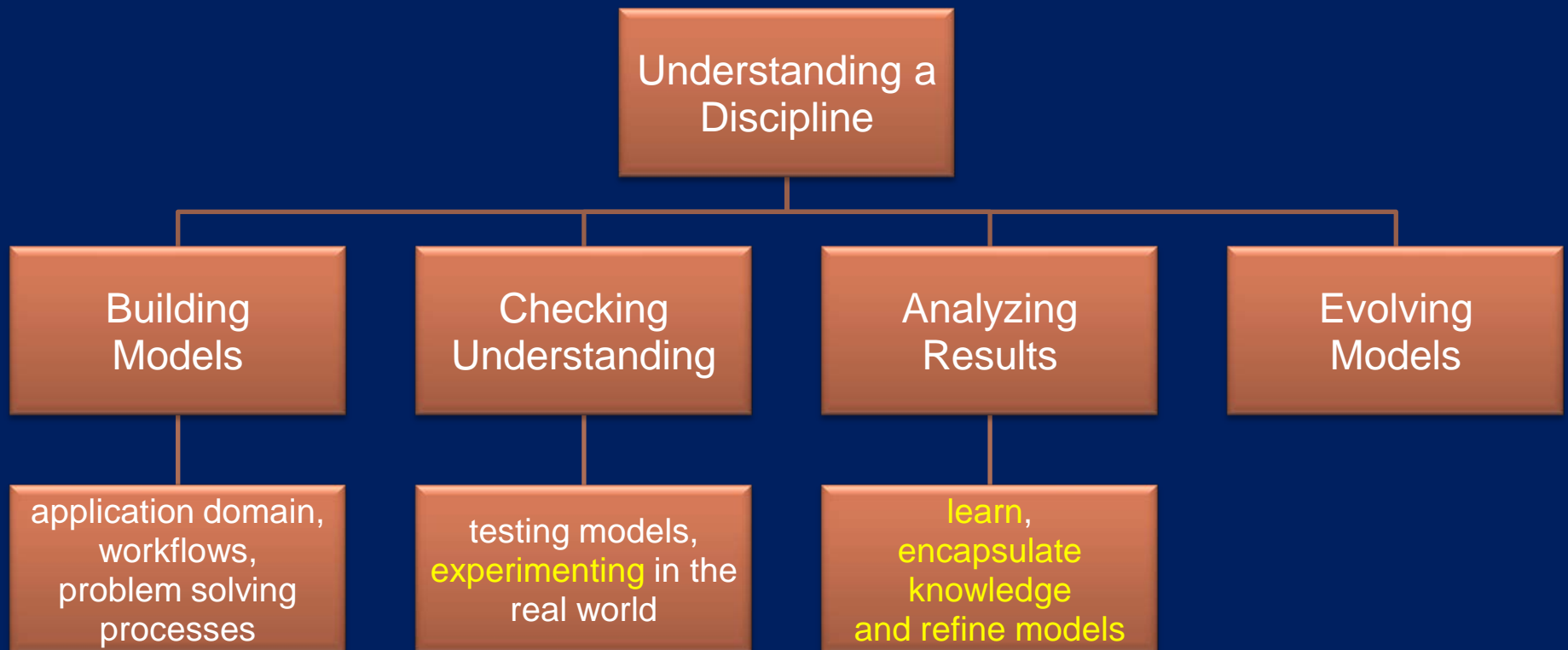


Defects

- Domain-specific defects?
- Can we identify patterns?
- Can we measure effort to find and fix defects?

Empirical Studies

Empirical → “Based on Observation”



Empirical Studies:

Answer Questions about a Technology

- Does it work better for certain types of people?
 - **Novices**: It's a good solution for training
 - **Experts**: Users need certain background knowledge...
- Does it work better for certain types of systems?
 - Static/dynamic aspects, complexity
 - Familiar/unfamiliar domains
- Does it work better in certain development environments?
 - Users [did/didn't] have the right documentation, knowledge, amount of time, etc... to use it

Empirical Studies: Types

Controlled Experiments

Typically:

- high cost
- small projects
- good basis for strong quantitative analysis and stronger statistical confidence in conclusions

Example

- comparing the effect of a new drug to a placebo

Case Studies

Typically:

- reasonable cost
- larger projects
- ability to simulate the effects of treatment variables in a realistic environment

Example

- Introducing a new development process and measuring the impact

Qualitative/Quantitative Analysis

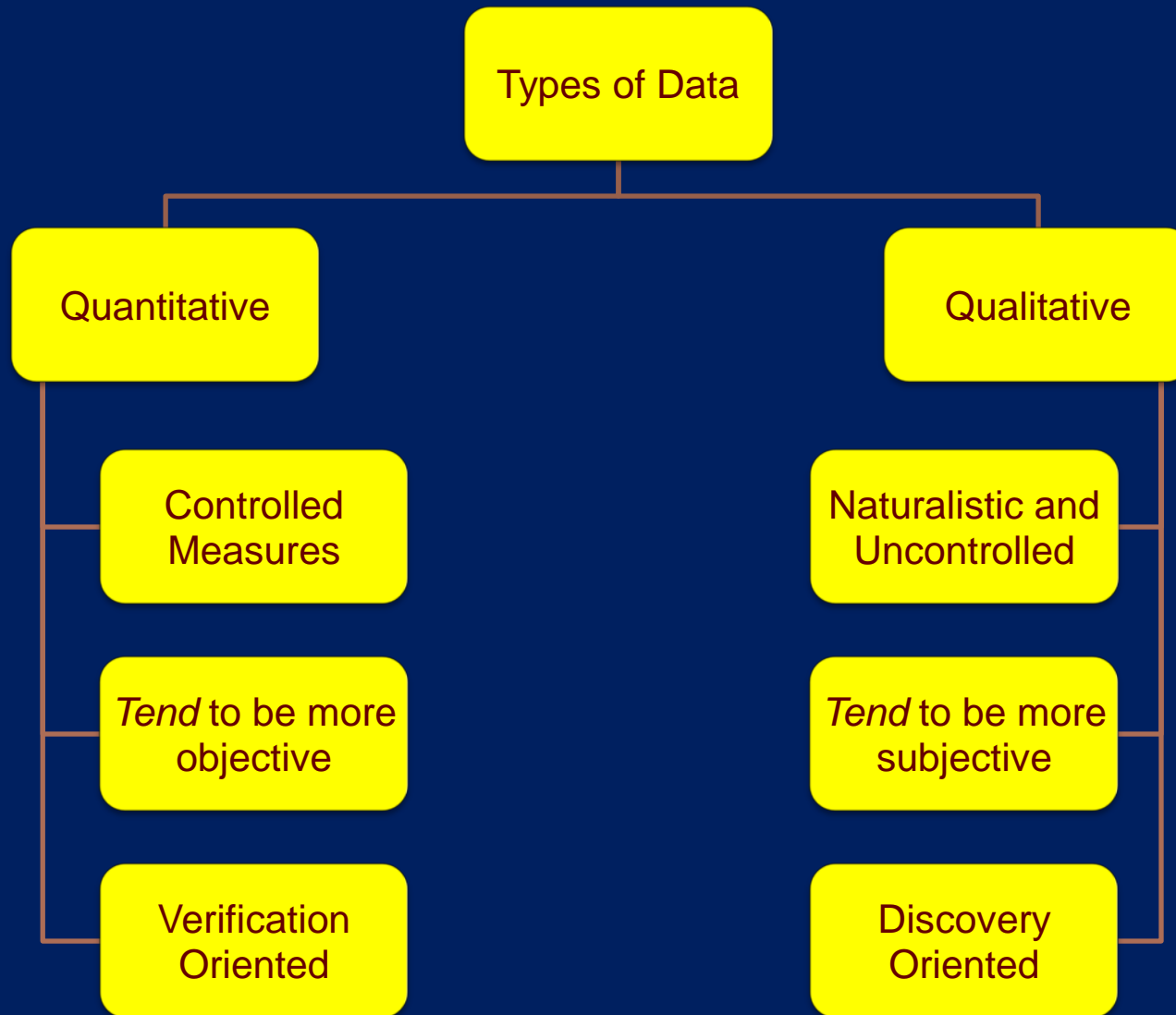
Typically:

- a set of variables for observation
- identified *a priori*

Example

- Observing professional developers to record their reaction to different types of defects in their software

Types of Data



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

Case Study Methodology

- Environment
 - Computational Science and Engineering projects
- Goals
 - Understand and document software development practices
 - Gather initial information about which 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

USERS: External; highly knowledgeable product engineers



LANGUAGE: OO-FORTRAN

CODE SIZE: ~405 KSLOC

TARGET PLATFORM:

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

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

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

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



LANGUAGE: FORTRAN77 (85%)

CODE SIZE: ~200 KSLOC

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

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

STAFFING: 3 FTEs



LANGUAGE: C++

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)

Projects Studied:

NENE

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

DURATION: ~25 Years

STAFFING: ~10 FTEs
(Thousands of contributors)

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



LANGUAGE: FORTRAN77
subset of FORTRAN90

CODE SIZE: 750 KSLOC

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

Projects Studied:

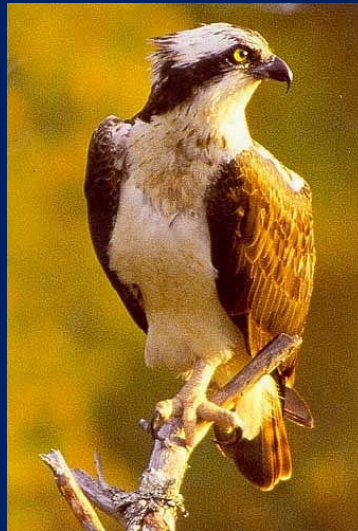
OSPREY

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)

STAFFING: ~10 FTEs

USERS: Hundreds of installations – some have hundreds of users



LANGUAGE: FORTRAN

CODE SIZE: 150 KLOC
(50 KLOC Comments)

TARGET PLATFORM: SGI,
IBM, HP, and Linux

Projects Studied: Summary

	FALCON	HAWK	CONDOR	EAGLE	NENE	OSPREY
Application Domain	Product Performance	Manufacturing	Product Performance	Signal Processing	Process Modeling	Weather Forecasting
Duration (Years)	~ 10	~ 6	~ 20	~ 3	~ 25	~10
# of Releases	9 (production)	1	7	1	?	?
Staffing (FTEs)	15	3	3-5	3	~10 (100's of contributors)	~10
Customers	< 50	10s	100s	None	~ 100,000	100s
Code Size (LOC)	~ 405,000	~ 134,000	~200,000	< 100,000	750,000	150,000
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
 - 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
- Need to develop methods that 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

- CS&E domain places more constraints on the language that traditional IT domain
- 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 -- 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**
 - Appropriate, flexible SE methodologies need to be employed for CS&E software development

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 and customers may come from different agencies
 - 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

Software Engineering for Computational Science and Engineering (SE-CSE) Workshops

SE-CSE Workshops

- Interaction between SE and CS&E
- Held at ICSE – Moving to ICCS (2010)
- Evolved from SE-HPC workshops
- Important Topics
 - **Differences** between research and IT software
 - CS&E software **quality goals**
 - Crossing the **communication** chasm
 - How to measure impact on **scientific productivity**

SE-CSE Workshops

Differences

- Complex domains
- Main focus on science/engineering
- Long lifecycles
- Investigation of unknown introduces risk
- Unique characteristics of developers
 - Deep knowledge of domain – lack formal SE
 - Often the main users of the software

SE-CSE Workshops

Quality Goals

- Lack of viable V&V techniques
- Focus on process transparency
- Guaranteed not to give an incorrect output
- Other SE characteristics not as important
 - Testability, reusability, maintainability

SE-CSE Workshops

Communication

- SE need to understand CS&E problems
- SE need to learn from CS&E developers
- Describe SE concepts in familiar terms
- Need people with expertise in both SE & CS&E
- CS&E teams have to realize a problem before needed help

SE-CSE Workshops

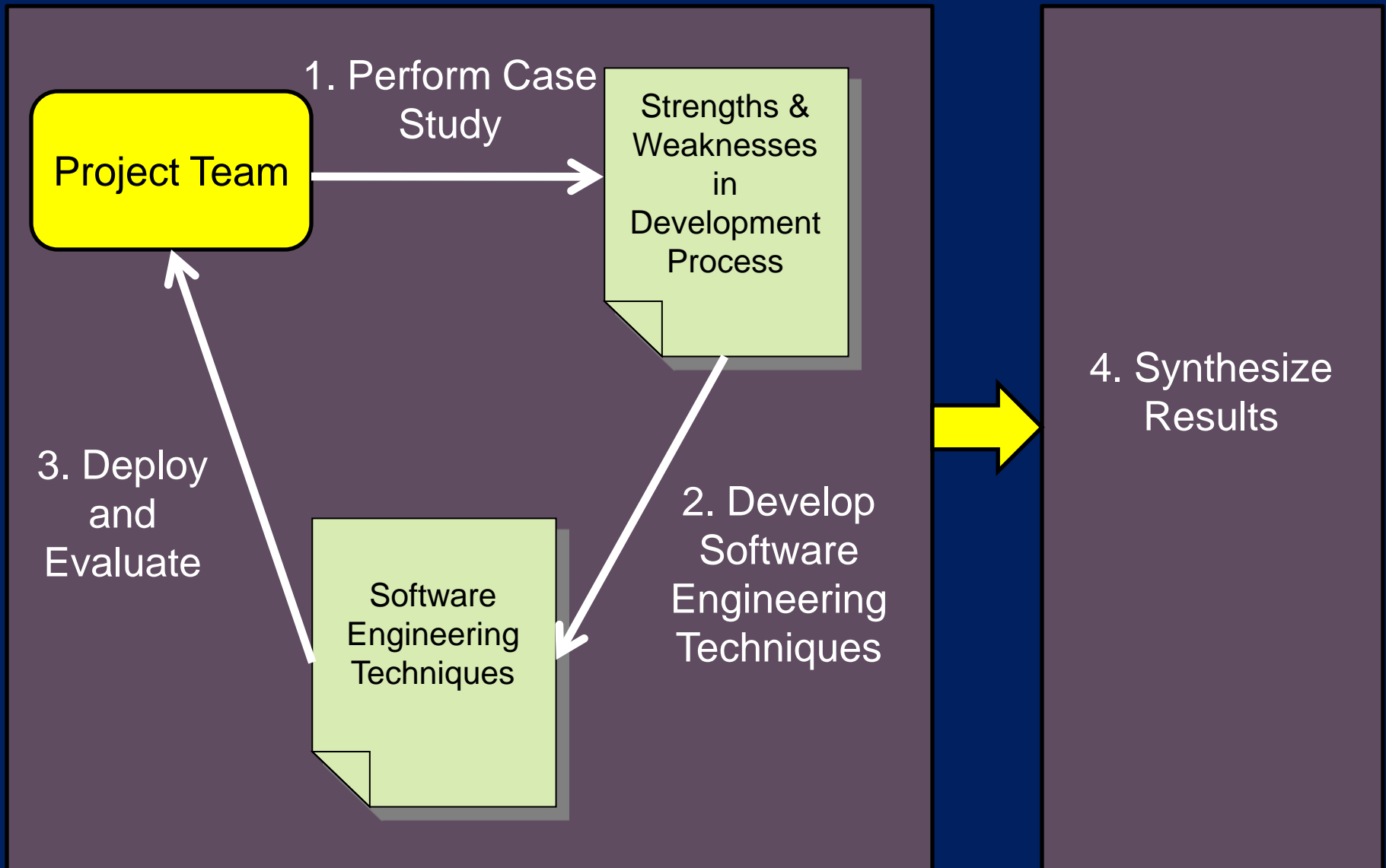
Scientific Productivity

- Need to evaluate impact
- Scientific productivity \neq Software productivity
- Need results in a relatively short time
 - Self-assessments
 - Word of mouth

Summary

- Five case studies of CS&E software projects
- Nine lessons learned
- Summary of SE-CSE workshops
- Contributions
 - Reasons why the development process is different for CS&E software
 - Insights into lack of use of traditional SE approaches
 - Ideas to guide the improvement SE for CS&E

Future Work – Collaboration Ideas



Acknowledgements

- Doug Post, Richard Kendall (LANL, SEI)
- Susan Squires (SUN)
- Christine Halverson (IBM)
- DARPA HPCS project

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 2007 International Conference on Software Engineering*. Minneapolis, MN. May 23-25, 2007. p. 550-559.
- Carver, J., Hochstein, L., Kendall, R., Nakamura, T. Zekowitz, M., Basili, V. and Post, D. " Observations about Software Development for High End Computing. *CTWatch Quarterly*. November, 2006. p. 33-37. (Invited Paper).
- Carver, J., Seaman, C., and Jeffery, R. "Using Qualitative Methods in Empirical Software Engineering." Lecture presented at *The International Advanced School on Empirical Software Engineering*. Redondo Beach, CA. 2004.
- Kendall, R.P., Carver, J., Mark, A., Post, D., Squires, S., and Shaffer, D. *Case Study of the Hawk Code Project*. Technical Report, LA-UR-05-9011. Los Alamos National Laboratories: 2005.
- 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.
- Kendall, R.P., Post, D., Squires, S., and Carver, J. *Case Study of the Eagle Code Project*. Technical Report, LA-UR-06-1092. Los Alamos National Laboratories: 2006.
- Post, D.E., Kendall, R.P., and Whitney, E. "Case study of the Falcon Project". In *Proceedings of Second International Workshop on Software Engineering for High Performance Computing Systems Applications (Held at ICSE 2005)*. St. Louis, USA. 2005. p. 22-26
- Shull, F. "Experimental Methods in Software Engineering." Lecture presented at *The International Advanced School on Empirical Software Engineering*. Redondo Beach, CA. 2004.

Thank You!

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

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