Lessons Learned from ASCI

Douglass Post

Los Alamos National Laboratory

International Workshop on Advanced Computational Materials Science for Nuclear Materials

Washington, DC

April 1, 2004

Abridged version of
LA-UR-04-0388
Approved for public release;
Distribution is unlimited

Los Alamos

Outline

- Introduction
- Case study “lessons learned”
- Quantitative Estimation
- Conclusions

Three Challenges
Performance, Programming and Prediction

- **Performance Challenge**-Building powerful computers
- **Programming Challenge**—Programming for Complex Computers
  - Rapid code development
  - Optimize codes for good performance
- **Prediction Challenge**—Developing **predictive** codes with complex scientific models
  - Develop codes that have reliable predictive capability
    - Verification
    - Validation
    - Code Project Management and Quality

Lessons Learned are important

- 4 stages of design maturity for a methodology to mature—Henry Petroski—*Design Paradigms*
- Suspension bridges—case studies of failures (and successes) were essential for reaching reliability and credibility

Tacoma Narrows Bridge buckled and fell 4 months after construction!

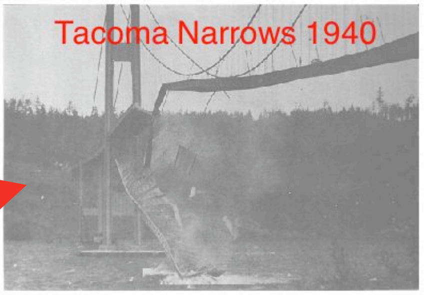
Case studies conducted after each crash. Lessons learned identified and adopted by community

time

1

2

3



Lessons Learned
Case Studies

4

March 30, 2004

Computational Science is at the third stage.

- Computational Science is in the midst of the third stage.
- Prior generations of code developers were deeply scared of failure, didn't trust the codes.
- New generation of code developers trained as computational scientists
- New codes are more complex and more ambitious but not as closely coupled to experiments and theory
  - Disasters occurring now
- We need to assess the successes and failures, develop "lessons learned" and adopt them
- Otherwise we will fail to fulfill the promise of computational science
- Computational science has to develop the same **professional integrity** as theoretical and experimental science

ASCI was launched to develop predictive capability for nuclear weapons

- In late 1996, the DOE launched the Accelerated Strategic Computing Initiative (ASCI) to develop the“enhanced” predictive capability by 2004 at LANL, LLNL and SNL that was required to certify the US nuclear stockpile without testing
  - ASCI codes were to have much better physics, better resolution and better materials data
  - Develop massively parallel platforms (20 TFlops at LANL this year, 100 TFlops at LLNL in 2005-2006)
  - ASCI included development of applications, development and analysis tools, massively parallel platforms, operating and networking systems and physics models
- ~ $6 B expended so far

Case Studies: *“Lessons Learned”*

The Successful ASCI projects emphasized:

- Building on successful code development history and prototypes
- Highly competent and motivated people in a good team
- Risk identification, management and mitigation Software Project Management: Run the code project like a project
- Determine the Schedule and resources from the requirements
- Customer focus
  - For code teams and for stakeholder support
- Better physics and computational mathematics is much more important than better “computer science”
- The use of modern but proven Computer Science techniques,
  - They don’t make the code project a Computer Science research project
- Develop the team
- Software Quality Engineering: Best Practices rather than Processes
- Validation and Verification

The unsuccessful ASCI projects didn’t emphasize these

LLNL and LANL had no “big code project” experience before ASCI

- But they did have a lot of successful small team experience—small teams developing and integrating one package at a time
- But ASCI needed rapid code development on an accelerated time scale
- LLNL and LANL launched large code projects with very mixed success
  - They hadn’t done it before and
  - They didn’t look at the “lessons learned” from other communities

It’s all about Teams!

- Tom DeMarco states that there are four essentials of good management:
  - “Get the right people
  - Match them to the right jobs
  - Keep them motivated.
  - Help their teams to jell and stay jelled.

 (All the rest is Administration.)”— “The Deadline”

Crestone ProjectTeam

- Management’s key role is the support and nurturing of teams

T. DeMarco, 2000; DeMarco and Lister, 1999; Cockburn and Highsmith, 2001; Thomsett, 2002; McBreen, 2001

Risk identification, management and mitigation are essential

- Tom DeMarco lists five major risks for software projects: T. DeMarco, 2002a
  1. Uncertain or rapidly changing Requirements, Goals and Deliverables
  2. Inadequate resources or schedule to meet the requirements
  3. Institutional turmoil, including lack of management support for code project team, rapid turnover, unstable computing environment, etc.
  4. Inadequate reserve and allowance for requirements creep and scope changes
  5. Poor Team performance.

  To these we add two:

  6. Inadequate support by stakeholder groups that need to supply essential modules, etc.
  7. Problem is too hard to be solved within existing constraints
- Poor team performance is usually blamed for problems
  – But, all risks but #5 are the responsibility of management!
- ASCI experience: Management attention to 1—4, 6,7 was been inadequate.

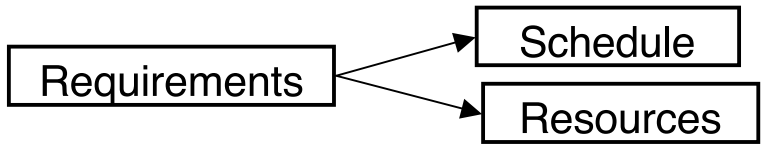
Software Project Management

- Good organization of the work is essential
- Manage the code project as a project
  - Clearly defined deliverables, a work breakdown structure for the tasks, a schedule and a plan tied to resources
- Execute the plan, monitor and track progress with quantitative metrics, re-deploy resources to keep the project on track as necessary
- Insist on support from sponsors and stakeholders
- Project leader must control the resources, otherwise the “leader” is just a “cheerleader”!

Brooks, 1987; Remer, 2000; Rifkin, 2002; Thomsett, 2002; Highsmith, 2001

Requirements, Schedule and Resources must be consistent.

- Requirements determine the schedule and resources



- Schedule
  - The rate of software development, like all knowledge based work, is limited by the speed that people can think, analyze problems and develop solutions.
- Resources
  - The size of the code team is determined by the number of people who can coordinate their work together
- **Specifying the schedule and/or resources plus the requirements has been one of the greatest problems for ASCI (and for many other code development projects in our experience).**
- **Then the code development plan is over-specified.**

D. Remer, 2000; T. Jones, 1988; Post and Kendal, 2002; S. Rifkin,2002

ASCI codes take about 8 years to develop.

- We have studied the record of most of the ASCI code projects at LANL and LLNL to identify the necessary resources and schedule
- The requirements are well known and fixed. LANL and LLNL have been modeling nuclear weapons for 50 to 60 years.
- The data indicate that it takes about 8 years and a team of at least 10 to 20 professionals to develop a code with the minimum capability to provide a 3 Dimensional simulation of a nuclear explosion

Post and Cook, 2000: Post and Kendall, 2002

We used simple metrics to estimate schedule and resource requirements

Key parameter is a "function point" —FP—, a weighted total of inputs, outputs, inquiries, logical files and interfaces

$$FP = \left(\frac{C++ \quad SLOC}{53} + \frac{C \quad SLOC}{128} + \frac{F77 \quad SLOC}{107} \right)$$

SLOC: Single Line of Code

$$Schedule = FP^x \quad 0.4 < x < 0.5; \quad use \quad x = .47$$

T. Capers-Jones, 1998

Corrections for Lab environment compared to industry

Schedule= FP schedule + delays

Delays up to 1.5 years for recruiting, clearance, learning curve

Schedule multiplier of 1.6 for complexity of project

1.6 based on contingency required compared to industry due to complexity of classified/unclassified computing environment, unstable and evolving ASCI platforms, paradigm shift to massively parallel computing, need for algorithm R&D, complexity of models, etc.

D. Remer, 2000; E. Yourdon, 1997

$$Team \quad Size = \frac{FP}{150} \qquad predicted \ bugs = FP^{1.25} \qquad predicted \ documentation = FP^{1.15}$$

ASCI codes take 8 years to develop

Software Resource Estimates for the LLNL and LANL Code Projects*

	LLNL			LANL Code Projects			
	ASCI A	ASCI B	Legacy A	Antero	Shavano	Blanca	Crestone
Single Lines of Code (1000s)	184k	490k	410k	300k	500k	200k	314k
Function Points	4800	4000	5400	2900	4800	3800	2900
estimated schedule	8.7	7.6	6.9	6.6	8.1	7.4	6.7
Project age (c. initial milestone)	3	9	N/A	4	3.5	8	8
Successful in achieving initial ASCI milestone	No	Yes	N/A	No	No	No	Yes
Estimated staff requirements	22	27	24	14	22	18	14
real team size	20	22	8	17	8	35	12

*Yellow shading denotes historical data; white background denotes computed numbers

Result of not estimating the time and resources needed to complete the requirements was failure of many code projects to meet their milestones.

- Code Projects were asked to do 8 years of work in 3.5 years
  - + Management changed the requirements 6 months before the due date
- Result:
  - 3 out of 5 code projects "failed" to meet the milestones (those with less than 8 years of experience)
  - 2 out of 5 code project succeeded in meeting the milestones (those with more than 8 years of experience)
- Management at LLNL and LANL blamed the code teams, and in some cases, punished them
  - Morale was devastated, project teams were reduced and are only now are beginning to recover
- **Real cause was no realistic planning and estimation by upper level management**
- DOE and labs now drafting more appropriate milestones and schedules

Post and Cook, 2000: Post and Kendall, 2002

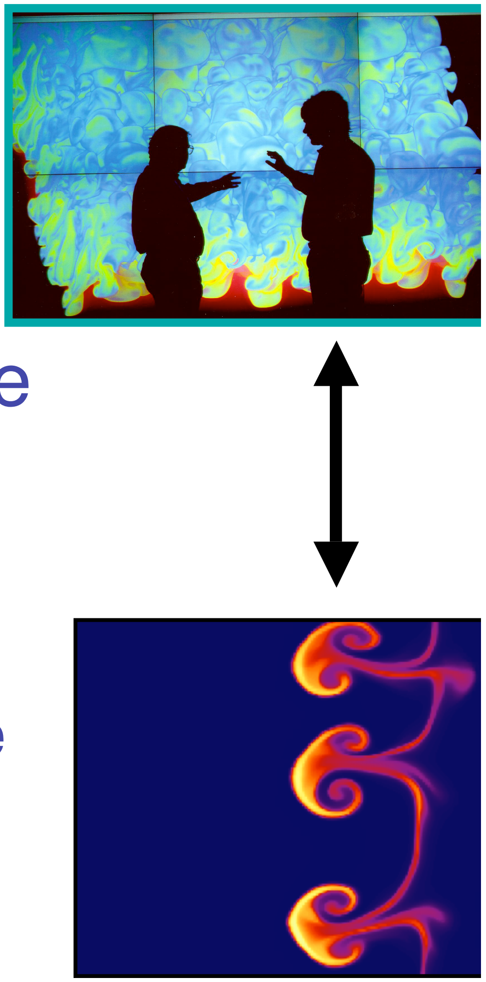
Focus on Customer

User

- Customer focus is a feature of every successful project
- Inadequate customer focus is a feature of many, if not most, unsuccessful projects
- Code projects are unsuccessful unless the users can solve real problems with their codes

McBreen, 2001; D. Phillips, 1997: R. Thomsett, 2002; E. Verzuh, 1999

Better Physics and math the key



- Predictive ability of codes depends on the quality of the physics and the solution algorithms
  – Correct and appropriate equations for the problem, physical data and models, accurate solutions of equations,…

R. Laughlin, 2002;Post and Kendall, 2002

Employ modern computer science techniques, but don’t do computer science research.

- Main value of the project is improved science (e.g. physics and math)
- Implementing improved physics (3-D, higher resolution, better algorithms, etc.) on the newest, largest massively parallel platforms that change every two years is challenging enough. Don’t increase the challenge!!!
- LANL spent over 50% of its code development resources on a project that had a major computer science research component. It was a massive failure (~$100M).

T. Demarco, T. Lister (2002); Post and Cook, 2000: Post and Kendall, 2002

Team Development is important!

- Codes are developed by teams, not by organizations or processes
- Invest in building the skills of the team members

Cockburn and Highsmith, 2001; DeMarco and Lister, 1999

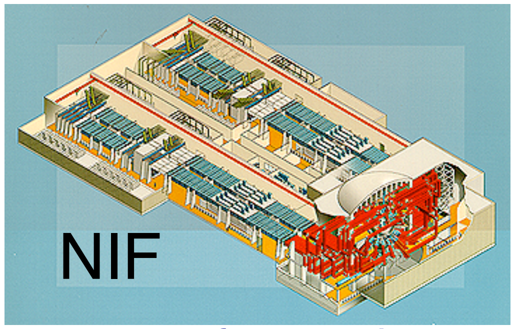
Software Quality Assurance: Practices rather than Processes

- Attention to project management is more important, but:
- Software Quality Assurance (SQA) and is a big issue for DOE and DOD codes (**10 CFR 830**, etc.)
- Heavy processes stifle innovative software—the kind of innovations necessary to solve computational scientific and technical problems
  - Scientists are trained to question authority, look for value added for any procedure
- ASCI had more success emphasizing “Best practices” than “Good processes”
- If the code team doesn’t implement SQA on their own terms, the sponsors may implement SQA on their terms, and the teams won’t like it much less

DeMarco and Boehm, 2002; DPhillips,1997; Remer, 2000; DeMarco and Lister, 2002; Rifkin, 2002; Post and Cook, 2000: Post and Kendall, 2002

Verification and Validation

- Customers (e.g. DOD) want to know why they should believe code results
- Codes are not reality, but only a model of reality
- Verification and Validation are essential
- Verification
  - Equations are solved correctly
  - Regression suites of test problems, convergence tests, manufactured solutions, analytic test problems, conserved quantities
  - New methods urgently needed
- Validation
  - Ensure models reflect nature
  - Check code results with experimental data
  - Need shift in experimental paradigms toward specific and dedicated validation experiments rather than use of existing data
  - NNSA is funding a large experimental program to provide validation data
    - National Ignition Facility, DAHRT, ATLAS, Z,…



DAHRT

Roach, 1998; Roache, 2002; Salari and Knupp, 2000; Lindl, 1998; Lewis, 1992; Laughliin, 2002)

Conclusions

- If Computational Science is to fulfill its promise for society, it will need to become as mature as theoretical and experimental methodologies.
- **Prediction Challenge**
  - Need to analyze past experiences, successes and failures, develop "lessons learned" and implement them—DARPA HPCS doing case studies of ~ 20 major US code projects (DoD, DOE, NASA, NOAA, academia, industry,…)
  - Major lesson is that we need to improve:
    - Verification
    - Validation
    - Software Project Management and Software Quality
- **Programming Challenge**
  - HPC community needs to reduce the difficulty of developing codes for modern platforms—DARPA HPCS developing new benchmarks, performance measurement methodologies, encouraging new development tools, etc.

Acknowledgements

I am grateful for the invitation from the program committee and for discussions with and assistance from my colleagues:

Tom Adams
Marvin Alme
Bill Archer
Donald Burton
Gary Carlson
John Cerutti
William Chandler
Randy Christiansen
Linnea Cook
Larry Cox
Tom DeMarco
Paul DuBois
Tom Gorman
Dale Henderson
Richard Kendall
Joseph Kindel
Kenneth Koch
Robert Lucas

Tom McAbee Douglas Miller
Pat Miller
Myasa Peterson
James Rathkopf
Bill Reed
Donald Remer
Anthony Scannapieco
Jamileh Soudah
David Tubbs
Robert Weaver
Daniel Weeks
Robert Weaver
Don Willerton
Dan Weeks
Ed Yourdon
Michael Zika
George Zimmerman.

References

1. Laughlin, R., The Physical Basis of Computability. Computing in Science and Engineering, 2002. 4(3): p. 27-30.
2. Petroski, H., Design Paradigms: Case Histories of Error and Judgement in Engineering. 1994, New York: Cambridge University Press. 221.
3. Gehman, H.W., et al., Report of the Columbia Accident Investigation Board. 2003, National Aeronautics and Space Administration: Washington, DC. p. 248.
4. Hallquist, J.O. Curent and Future Developments of LS-DYNA-1. in 4th European LS-DYNA Conference. 2003. ULM, Germany: Livermore Software Technology Corporation.
5. Taleyarkhan, R.P., et al., Evidence for Nuclear Emissions During Acoustic Cavitation. Science, 2002. 295(1): p. 1868-1873.
6. Shapira, D. and M. Saltmarsh, Nuclear Fusion in Collapsing Bubbles—Is It There? An Attempt to Repeat the Observation of Nuclear Emissions from Sonoluminescence. Physical Review Letters, 2002. 89(10): p. 104302-104305.
7. Post, D. and R. Kendall. Lessons Learned From ASCI. in DOE Software Quality Forum 2003. 2003. Washington, DC: Los Alamos National Laboratory.
8. Thomsett, R., Radical Project Management. 2002, Upper Saddle River, NJ: Prentice Hall.
9. DeMarco, T., The Deadline. 1997, New York, New York: Dorset House Publishing. 310.
10. Beck, K., Extreme Programming Explained. 2000, Boston: Addison Wesley.
11. Remer, D. Managing Software Projects. in UCLA Technical Management Institute. 2000. Los Angeles, CA: UCLA Extension Courses.
12. Vliet, H.v., Software Engineering, Principles and Practice. 2000, Chichester: John Wiley and Sons, Ltd. 726.
13. Brooks, F., The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition. 1995, Menlo Park: Addision-Wesley Publishing Co. 322.
14. Verzuh, E., The Fast forward MBA in Project Management. 1999: John Wiley.
15. Ruskin, A.M. and W.E. Estes, What Every Engineer Should Know About Project Management. 2 ed. What Every Engineer Should Know, ed. W. H.Middendorf. Vol. 33. 1995, New York: Marcel Dekker, Inc. 274.
16. DeMarco, T. and T. Lister, Waltzing with Bears, Managing Risk on Software Projects. 2003, New York, New York: Dorset House Publishing. 196.
17. Glass, R.L., Software Runaways: Monumental Software Disasters. 1998, New York: Prentice Hall PTR. 288.
18. Demarco, T. and T. Lister, Risk Management for Software. 2002, The Cutter Consortium: Arlington, MA.
19. Capers-Jones, T., Estimating Software Costs. 1998, New York: McGraw-Hill.
20. Yourdon, E., Death March. 1997, Upper Saddle River, NJ: Prentice Hall PTR.
21. Symons, C.R., Function Point Analysis: Difficulties and Improvements. IEEE Transactions on Softwware Engineering, 1988. 14(1): p. 2-11.
22. Brooks, F.P., No Silver Bullet: Essence and Accidents of Software Engineering. Computer, 1987. 20(4): p. 10-19.
23. Oberkampf, W. and T. Trucano, Verification and Validation in computational fluid mechanics. Progress in Aerospace Studies, 2002. 38: p. 209-272.
24. Roache, P.J., Verification and Validation in Computational Science and Engineering. 1998, Albuquerque: Hermosa Publishers. 446.
25. Roache, P.J., Code Verification by the Method of Manufactured Solutions. Transactions of the ASME, 2002. 124: p. 4-10.
26. Kamm, J.R., W.J. Rider, and J.S. Brock. Combined Space and Time Convergence Analysis of a Compressible Flow Algorithm. in AIAA Conference on Computational Fluid Dynamics. 2003. Orlando, Florida: AIAA.
27. Pautz, S.D. Verification of Transport Codes by the Method of Manufactured Solutions: the ATTILA Experience. in ANS International Meeting on Mathematical Methods for Nuclear Applications. 2001. Salt Lake City, Utah: American Nuclear Society.
28. Paulk, M., The Capability Maturity Model. 1994, New York: Addison-Wesley.
29. Halberstam, D., The Reckoning. 1986, New York: William Morrow and Co.
30. Phillips, D., The Software Project Manager's Handbook. 1997, Los Alamitos: IEEE Computer Society.
31. DeMarco, T. and B. Boehm, The Agile Methods Fray. Computer, 2002. 35(6): p. 90-92.
32. Boehm, B., Get ready for agile methods, with care. Computer, 2002. 35(1): p. 64-69.
33. Highsmith, J. and A. Cockburn, Agile software development: the business of innovation. Computer, 2001. 34(9): p. 120-127.
34. Herbsleb, J., et al., Software Quality and the Capability Maturity Model. Communications of the ACM, 1997. 40(June, 1997): p. 30-40.
35. Humphrey, W.S., Winning with Software: An Executive Strategy. 2001, Pittsburg: Software Engineering Institute.
36. Huizenga, J., et al., Cold Fusion Research, A Report of the Energy Research Advisory Board to the United States Department of Energy. 1989, United States Department of Energy: Washington, DC 20585. p. 62.