Software Engineering Lessons Learned

Douglass Post

Los Alamos National Laboratory

Fall Creek Falls Conference

Fall Creek Falls, Tennessee, Oct. 19-20, 2004

LA-UR-04-4151

Approved for public release;
Distribution is unlimited

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Los Alamos

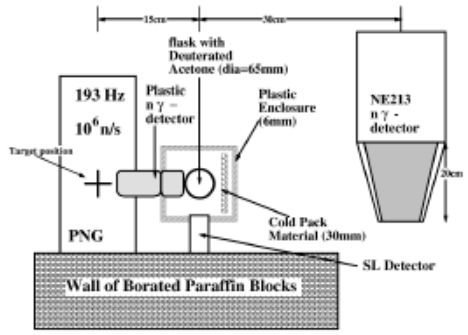
Computational Science faces Three Challenges.

- **Performance Challenge**—large, but progressing
  - Building high performance computers—big challenge
  - Price of performance is complexity
- **Programming Challenge**—larger, but progressing
  - Programming for high performance, complex computers—bigger challenge
  - Next generation of HPCs : 100,000's of processors
- **Prediction Challenge***—largest, but not as much progress
  - Develop complex, multi-effect simulations that have sufficient reliability that they can for the basis of important decisions—biggest challenge

*The Coming Crisis in Computational Science, Proceedings IEEE International Conference on High Performance Computer Architectures, Madrid, Spain, February 15, 2004

Computational predictions aren't yet reliable.

- **An ORNL example:**
- Taleyarkhan and co-workers at ORNL used intense sound waves to form sonoluminescent bubbles with deuterated acetone
- They observed Tritium formation and 14 MeV neutrons, indicating that nuclear fusion was occurring
- "Observed" Temperature was ~ 10^7 ^oK instead of the usual 10^3 ^oK
- If true, we could have a fusion reactor in every house
- Computer simulations were done that matched the "experimental results" if the driving pressure in the codes was increased by a factor of 10 (well outside reasonable uncertainties)
- Based on the "agreement" of "theory" and "experiment", the results were published in Science and generated intense interest
- Experiments were repeated, especially at ORNL. No significant Tritium or 14 MeV neutrons were found.
- "Sigh" — No fusion reactor in everyone's house.
- Simulation was misleading
- Bad experiment + bad simulation = ?

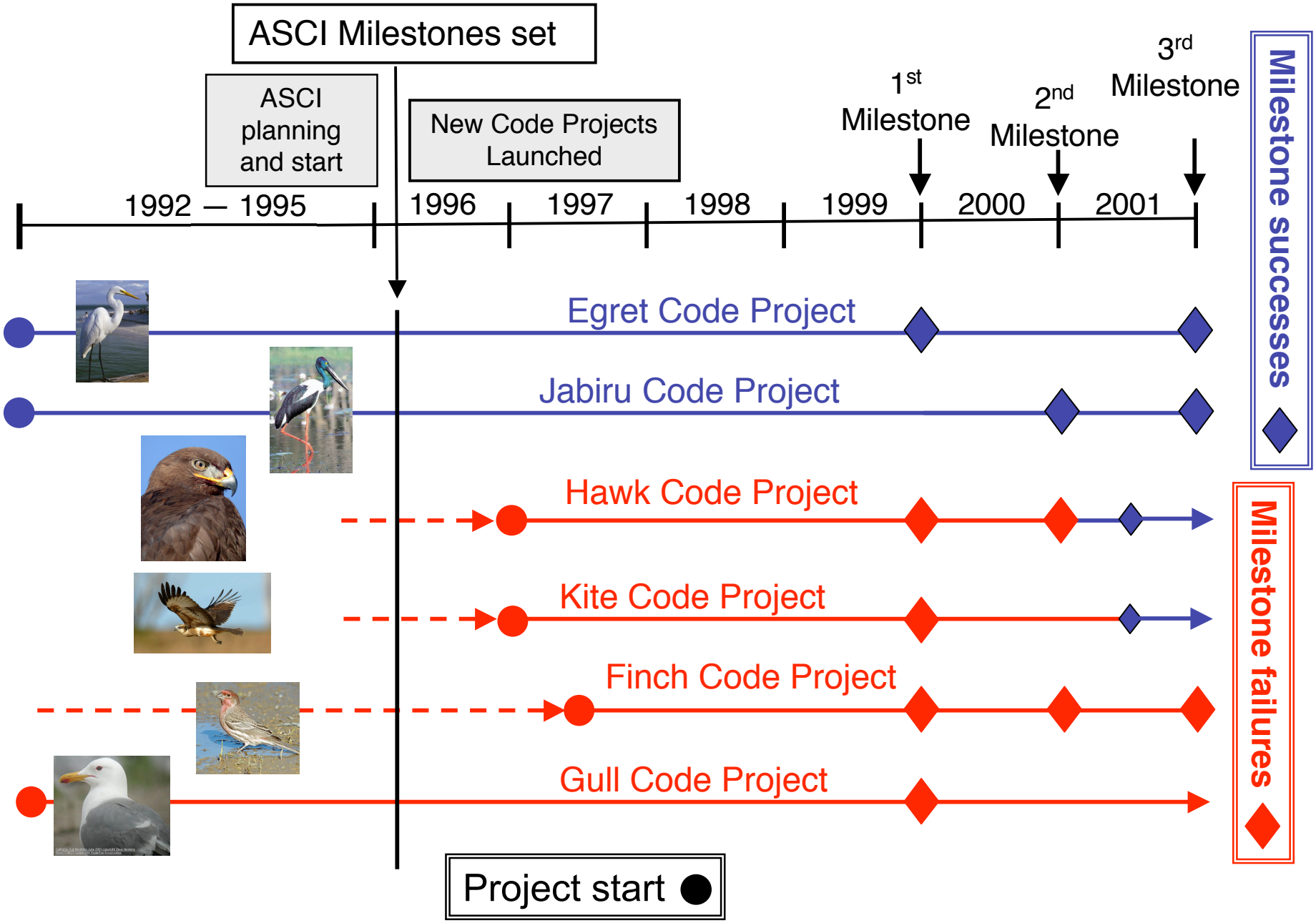


—Taleyarkhan et al, Science **295**(2002), p. 1868; Shapira, et al, PRL, **89**(2002), p.104302.

ASCI Successes and Failures

- 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
  - ~ $6 B expended so far (included codes, platforms, and supporting software infrastructure)
  - Six projects were launched in 1996 and 1997
  - Milestones were set in late 1996 by senior management, milestones were set for 3, 4 and 5 years later
- Success required development and integration of several major physics capabilities
  - Takes 8 years to develop a 3-D nuclear explosion code.
  - 2/3 of projects failed to meet the first milestones
  - 1/3 have been successful, 1/3 are on the road to success, and 1/3 were eventually canceled
  - 1/2 of the code projects had 3 to 4 years to do 8 years of work, and none of them could do it
- **A 33 to 67% failure rate, we must do better!!!!!!!**
- Based on our case studies and other evidence, DOE, LLNL and LANL recognized this and have developed more realistic goals and schedules

ASCI Code Project Schedule



We studied these projects to identify the *"Lessons Learned*"*

The Successful projects emphasized:

- Conservative approach
  - Building on successful code development history and prototypes
  - Better physics and computational mathematics over better "computer science"
  - The use of modern but proven Computer Science techniques,
    - Don't make the code project a Computer Science research project!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
- Sound Software Project Management
  - Highly competent and motivated people in a good team
  - Development of the team
  - Software Project Management: Run the code project like a project
  - Determining the Schedule and resources from the requirements
  - Identifying, managing and mitigating risks
  - Focusing on the customer
    - For code teams and for stakeholder support
  - Software Quality Engineering: Best Practices rather than Processes
- Verification and Validation
  - Need for improved V&V methods became very apparent

The unsuccessful projects didn't emphasize these!

*Lessons Learned from ASCI, DE Post, RP Kendall, International Journal of High Performance Computing Applications, to appear, Fall, 2004.

Lessons Learned are the way forward!!!

- 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
- Computational Science is at stage 3

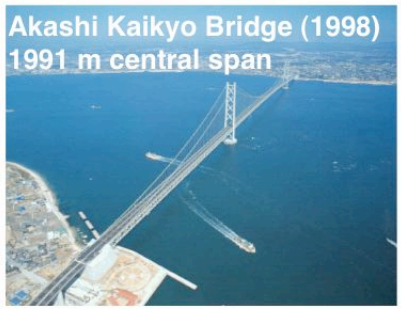
1

2

3

Lessons Learned
Case Studies

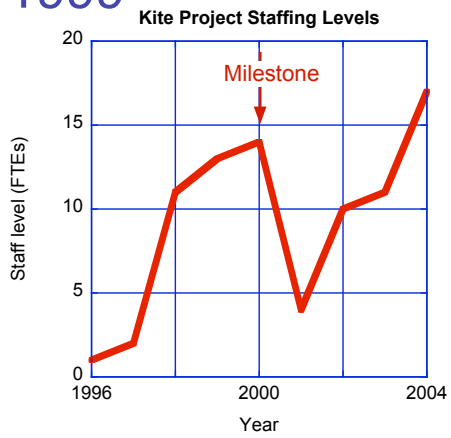
4



time

Kite Code Project had initial difficulties, but is now headed for success

- Project launched in mid-1996 with innovative, promising but feasible physics approach
- Requirements and goals set by sponsors without much input from customers
  - Milestones were set without adequate consideration of consistency among requirements, schedule and resources.
  - KITE project had to accomplish in 3 years what took the other teams 8 years.
  - Milestone problems bore little relation to needs of customers, were viewed by customers as “stunts”, result was no support from customers
    - Kite project didn’t complete milestone on time Dec. 31, 1999
    - Missing the milestone resulted in staffing cut
- After milestone failure, project was refocused on customer needs, customers set requirements, customers began to support project



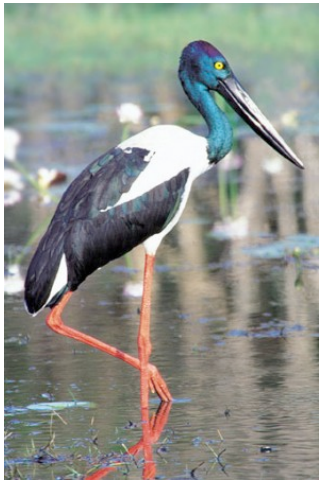
Important Lessons were identified.

- Project now back to healthy staff levels and is succeeding by focusing on customer support
- Some Lessons learned:
  - Experienced and knowledgeable people are essential for setting requirements, schedule and resource allocations
  - Realistic schedule is crucial, Kite team had 3 years to accomplish a 8 project
  - Failure to complete initial milestone led to punishing Kite team for not completing 8 years of work in 3 years, team and project almost destroyed
  - Customer requirements are more important than sponsor requirements
  - **Requirements management is crucial.**
  - Computer science support is important! Physicists are not the best people for programming and computer science issues (Programming Challenge)

Egret Code Project was conservative—schedule and technology—and successful

- Originally started in 1992 as part of a graduate thesis building on a prior code
  - Serial version working with all but one of the major required packages in 1994, parallelization began in 1997
- Written in C by a very experienced team of physicists and computer scientists
  - Management experienced with code development
- Early customer involvement (~1995), fairly continuous V&V since 1996
- Began implementation of next major package in 1997
- Strong support by management, well-defined project structure
- Completed first major milestone Dec 28, 1999, three days before Jan. 1, 2000 milestone due date

Jabiru code project was conservative and successful

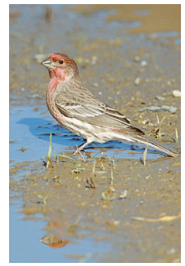


- Jabiru code project began as a code port from a vendor with all but one of necessary packages (serial) in 1992
  - Installation of other packages begun in 1997
  - Parallelization started in 1994
- Written in Fortran 90
  - Heavy emphasis on portability, (~ 10 different platforms)
  - Conservative, perhaps overly conservative computer science
- Continuous interaction and use by users, long V&V history
- Substantial use by non-ASCI projects (Labs, universities, and other institutions)
- Senior management support polarized, some strongly in favor—"it really works", some strongly opposed ("not invented here syndrome")
- Strong example of a "hero" model success, highly experienced and mature staff

Hawk Code Project was very ambitious.

- The Hawk Code Project Vision involved using object oriented languages(C++) with a Python controller
- Project began in late 1996
- Very few experienced staff, all very bright but fairly new to the physics and the programming challenge
  - Leadership very young and inexperienced, but smart and dedicated
  - Lots of computer science and programming support
  - Senior management very inexperienced managing code development
- Ambitious physics scope, trying new algorithms that needed considerable R&D
- Initially didn't succeed in meeting milestones—initial milestone not important for customers—customers weren't interested
- Still not a really mature code after 7 years
- Performance a major issue (10 times slower than comparable F90 codes)

Finch Code Project was a shotgun marriage that ended in divorce



- Finch Code Project began in early 1997
- Initial project consisted of two major groups in other divisions with existing codes that had lost their support plus an integration group
  - These groups viewed ASCI as a way to get support for their existing work
  - Success was measured by the success of each sub-project individual rather than the success of the whole project
  - The senior management in the other divisions was much more interested in the continued flow of the money than in the success of the Finch project
  - Little or no analysis was done to establish that the two packages were compatible, i.e. that it was technically feasible to integrate the two packages into a single code
  - Finch Project manager had almost no authority over the staff in each sub-project
- The Finch Project never really got off the ground.
  - It didn't meet any of its milestones
  - But it had the best SQA of all of the codes!
- The Finch Project was canceled last year.

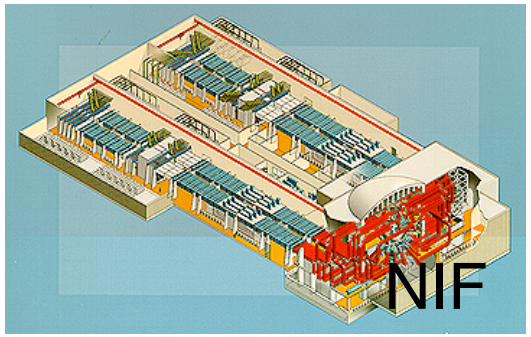
Gull Code Project was canceled after ~$100M

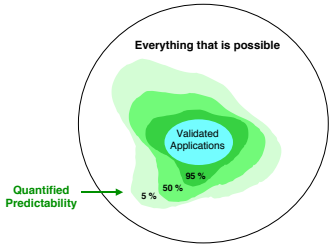
- The Gull Code Project was begun in 1992
- It was based on a F90 parallel code obtained from another laboratory
- The Gull project staff decided to use the project as a vehicle for improving code development methodologies
  - Automatic and general parallelization through a library
  - Latest object oriented computer languages (C++)
  - Automatic differencing and code generation from “equations”
  - Performance issues: Substantially outperformed by legacy code that it was based on (which only had about 1 FTE of support per year)

- Very large code team (~ 50 staff) spread across 3 divisions without much authority given to project lead
  - Project leaders were often inexperienced and there was substantial project leadership and staff turnover
- Hostility toward customers (“We know what you need better than you do!”)
- Failed to meet ultimate milestones (although a few intermediate ones were met)
- Canceled last year after $100M (about expending about one-half of the weapons code development resources at one of the labs)

Verification and Validation

- Customers (e.g. DOD for ASCI) want to know why they should believe code results
- Codes are only a model of reality
- Verification and Validation are essential
- Verification
  - Verify equations are solved correctly
  - Regression suites of test problems, convergence tests, manufactured solutions, analytic test problems, code comparisons and benchmarks
- Validation
  - Ensure models reflect nature, check code results with experimental data
  - Specific validation experiments are required
    - NNSA is funding a large experimental program to provide validation data—e.g. National Ignition Facility, DAHRT, ATLAS, Z,…
- V&V experience with ASCI codes indicates that a stronger intellectual basis is needed for V&V
- More intense efforts are needed in both types of V&V if computational science is to be credible





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 must 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.