

# Case Study of the Falcon Code Project

D.E. Post  
Los Alamos National Laboratory  
P.O. Box 1663, MS E526  
Los Alamos, NM 87544  
1-505-665-7680  
post@ieee.org

R.P. Kendall  
Los Alamos National Laboratory  
P.O. Box 1663, MS B260  
Los Alamos, NM 87544  
1-505-665-0356  
rpk@lanl.gov

E.M. Whitney  
Los Alamos National Laboratory  
P.O. Box 1663, MS C920  
Los Alamos, NM 87544  
1-505-667-3595  
whitney@lanl.gov

## ABSTRACT

The field of computational science is growing rapidly. Yet there have been few detailed studies of the development processes for high performance computing applications. As part of the High Productivity Computing Systems (HPCS) program we are conducting a series of case studies of representative computational science projects to identify the steps involved in developing such applications, including the life cycle, workflows and tasks, and technical and organizational challenges. We are seeking to identify how software development tools are used and the enhancements that would increase the productivity of code developers. The studies are also designed to develop a set of “lessons learned” that can be transferred to the general computational science community to improve the code development process. We have carried a detailed study of the Falcon (Fig.1) code project. That project is located at a large institution under contract to a national sponsor. The project team consisted of about 15 scientists charged with developing a multi-physics simulation that would utilize large-scale supercomputers with 1000s of processors. The expected life time of the code project is about 30 years. The case study findings reinforced the importance of sound software project management and the challenges associated with verification and validation.

## Categories and Subject Descriptors

D.2.0 [Software Engineering]. D.2.9 [Management]: *Life cycle, Productivity.*

## General Terms

Management, Verification

## Keywords

High Performance Computing, Verification And Validation, Software Project Management, Case Studies

## 1. INTRODUCTION

Computational Science potentially offers an unprecedented ability to predict the behavior of complex system and thus address many important societal issues. It has sometimes been described as the third leg of the triad of theory, experiment and simulation. However it is still relatively immature as a problem solving methodology compared to theory and experiment. Other problem

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies

bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SE-HPCS'05, May 15, 2005, St. Louis, Missouri, USA.

Copyright 2005 ACM 1-59593-117-1/05/0005...\$5.00..

solving procedures have matured as a result of learning from their past experiences, and by identifying and applying “lessons learned” from their successes and failures[1].



Figure 1. Falcon in flight.(Lanner Falcon - *Falco biarmicus*)

Through the DARPA High Productivity Computing Systems (HPCS) program, we are doing a set of case studies of large-scale high performance computing application code projects to develop these “lessons learned.” Additionally, these case studies will be used to develop an understanding of the scale of these projects and the challenges and tasks that code developers and code users face. This information will be used to help computer and software vendors focus on issues such as the required improvement for software development tools that must be addressed if computational science is to become more productive. As is common in social science case studies[2], we maintain the anonymity of the code project, the institution, and the sponsoring organization. Falcon is a pseudonym. We have found that anonymity is crucial if we are to obtain accurate information from the code project teams.

Section 2 of this paper provides an overview of the characteristics of the Falcon project. Section 3 focuses on the project team, the structure of the project, and the organization. Section 4 discusses the life cycle. Section 5 describes the workflow, the development tasks, who accomplishes them and how long they take. This section includes an assessment of the software development tool use and experience. Section 6 summarizes the “lessons learned”.

## 2. Code Characteristics

The goal of the Falcon code project is to develop a predictive capability for a product whose performance involves the trade-off

of many strongly coupled physical effects spanning at least ten orders of magnitude each of temporal and spatial scales. An accurate predictive capability is needed to reduce the dependence of the sponsoring institution on large, expensive and potentially dangerous empirical tests to certify the product.

The Falcon code project is based on an innovative and potentially very powerful method for solving a set of initial value partial differential equations for the conservation of particles, momentum and energy. These equations are non-linear and have non-linear source terms and coefficients that are calculated with analytic, computational and table look-up schemes. The coupled set of equations is solved with operator splitting, and some degree of time and spatial error correction. A mixture of explicit and implicit techniques is used. The Falcon code was designed to run on massively parallel SMP platforms. The product to be simulated is a multi-material object with a complicated geometry. The equations are solved on an unstructured 2 or 3 dimensional mesh that represents the major features of the object. Generating a reliable mesh from CAD-CAM files and other descriptions of the problem is a highly challenging task, often consuming several months of an expert's time to set up each new type of problem. The unstructured mesh allows flexibility for incorporating adaptive mesh refinement and adding resolution for capturing fine scale features where necessary.

Parallelization for computation with SMP architectures is accomplished with domain decomposition of the mesh using ParMetis[3]. The parallel programming model is MPI[3]. The target platforms are a SMP LINUX cluster with ~1000 nodes and a large vendor-specific SMP cluster with approximately 2000 nodes.

The approach to performance optimization is pragmatic. The team uses several optimization tools (e.g. PIXIE, DCPI, SpeedShop and prof[3]) to identify roadblocks. The team then works on minimizing the impact of the roadblocks. The emphasis during the early stages of development was to maximize performance through reasonable choices for the code architecture, and then to work on optimization after the basic capability of the code had been established. This approach was a response to the pressures to develop the basic capability necessary for demonstrating the actual and potential utility of the code as soon as possible, even if the initial performance efficiency is low. If the required capability had not been demonstrated in a timely fashion, the project would have been canceled. A substantial investment in optimization was thus a luxury to be addressed at a later time.

The code project uses nine different languages and a set of external libraries including: Fortran, C, Perl, Python, Unix shells, SCHEME, MAKE, and external libraries. Most of the code is an object-oriented instantiation of Fortran. The team has successfully captured many of the advantages of low level object-oriented capability, such as polymorphism and inheritance, while avoiding the pitfalls of many levels of inheritance and excessive use of templating. The major blocks of code are about 410,000 Fortran SLOC, 50,000 SLOC of C, 200,000 SLOC of library code, and about total 30,000 SLOC of Perl, Python and Unix scripts. Perl and Python are primarily used for build and test scripts.

The Falcon project computational tools are used by a team of approximately 50 engineers to assess the behavior of new and existing product designs. The users are highly knowledgeable and experienced. They do most of the validation of the code by comparing the code results with data from past experiments and a

few new experiments. Their level of experience and expertise is sufficiently high that they can not only identify when bugs and model deficiencies are present but can often identify the source of the bug or the needed model improvements. The users participate very constructively and effectively in the development, verification and validation of the code. The code is extensively documented on an internal web-site (approximately 400 Mbytes of HTML files). The documentation consists of descriptions of the physics in the code, the algorithms and models in the code, the input and output, and instructions for how to run the code. This has proved highly useful for the users.

### 3. Code Project and Team Characteristics

As the Falcon project has grown, the formality of the management approach has increased. The project leader is in charge of general coordination of planning, monitoring and directing the work. He has delegated direct oversight of the work to "activity coordinators" who are responsible for the technical sub-tasks, including not only physics and algorithm issues, but also software quality assurance, testing, configuration management and documentation. These tasks are monitored by informal reports at weekly team coordination meetings and are tracked formally with quarterly reports. This structure allows detailed tracking and monitoring of tasks, while encouraging staff development. The whole team has access to the plans and status.

The Falcon project team consists of about twelve physicists and five computer scientists. It is highly trained, cohesive and competent. Team building has been emphasized. Attention has been given to succession planning and recruiting, both by the team and by management. Mentoring of new staff by experienced staff is emphasized. Many of the project team will typically work on a single project like Falcon for 10 to 20 years or more.

The Falcon project had a difficult initial phase (Fig. 3). The sponsor and senior institutional managers specified the original requirements and the project schedule. The initial schedule was too ambitious and the original requirements didn't meet the needs of the users. Neither the sponsor nor institutional management had much experience in the area being addressed by the Falcon Project. They didn't rely heavily on the substantial body of experience that had been developed in earlier projects similar to Falcon. The original requirements and schedule called for the development of an initial capability three years after project start, but the historical record and quantitative estimates[4] indicate that it normally takes approximately eight years to develop this level of initial capability. As a result, the team didn't complete its first milestone on schedule (i.e. accomplish eight years of work in three years). A further complication was that the milestones were not aligned with the requirements and needs of the users. The users viewed the milestones as "stunts" that had little to do with addressing the issues they judged important. The team staffing level was reduced from fifteen to about four immediately following the missed milestone (Fig. 2).

While this was painful for the code team, it had the advantage of taking the code out of the management spotlight. The code team was then able to concentrate on addressing the needs and interests of the user community, and was able to develop a strong connection to the users. This led to initial acceptance and later to strong advocacy of the code project by the user community. With this advocacy, senior management then began to support the code project. New staff were recruited and the project is now a key part of the institution's and sponsoring agency's program.

As a result of the customer advocacy and an analysis by middle management, the project requirements were modified to closely match the requirements of the users and a more realistic schedule was developed.

We judged the level of maturity of the code project to be somewhere between CMM Level 2 and Level 3 in terms of the processes and practices the Falcon code group is following[5]. The team has not had a formal CMM assessment, but has had several internal and external audits.

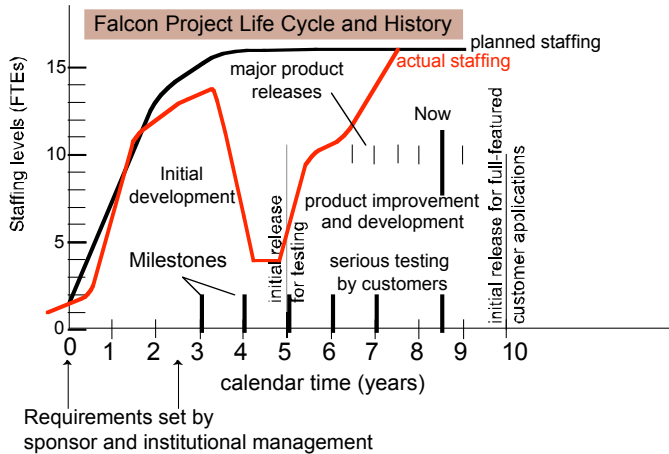


Figure 2. FALCON code project staffing and release schedule.

#### 4. Falcon Life Cycle

The FALCON code project lifetime is expected to be on the order of 30 years (Figure 3). This is based on the experience with similar projects at this institution. Indeed, some projects like Falcon have had lifetimes of up to 45 years. The first part of the life cycle was dedicated to development of the preliminary, initial capability to solve the conservation equations without accurate source terms or coefficients. This took about five years. Now that capability is being tested. Further development will continue until a production capability has been achieved with more accurate source terms and coefficients. The production phase involves heavy use and testing by the user community. During the production phase, the code team will support the use of the code, maintain the code, port it to new platforms, and develop and add

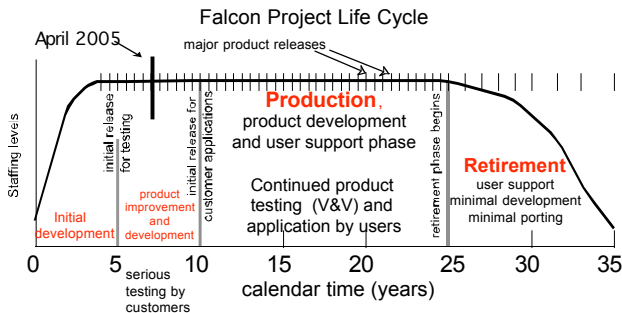


Figure 3. Falcon Project Life Cycle. The small tic marks denote 6 month release dates.

new capability as required by product engineers.

For similar projects at this institution, the ultimate life span of the code is determined by user demand and the difficulty of

successively porting the code to new platforms. When a successor can replace the older code, and the product engineers have made the transition from the older code to the successor code, support for the older code is stopped and it is “retired”. The development of new capability then shifts to the successor code. The Falcon project is in the process of displacing an older project with less capability. The life-time of these projects is much longer than the time between new platforms. Thus porting to the new platform becomes much more important than extensive performance optimization for a particular platform.

Like many computational simulations, the FALCON code project has a strong element of research and development to ensure that new algorithms are developed and successfully implemented. The users also have needs that must be met if the code project is to be successful. The adequacy of the models in the code can only be determined as part of an intensive validation program. It was difficult to draft a detailed list of requirements before the project was begun or to specify a detailed schedule.

In the case of the Falcon project, senior institutional management and the sponsor specified a set of requirements that would allow them to “sell” the program to the funding sources. This is similar to experiences in the Information Technology (IT) industry where a marketing department identifies market opportunities, and then signs up customers by promising a level of code capability that outbids the competition. Then the software engineers must deliver the promised capability. This contributes to over-promising the capability that can be delivered within the defined schedule and resource level[6, 7].

In the case of the Falcon project, the detailed schedule initially specified by the sponsor and senior institutional management was not based on the prior experience with similar codes or quantitative estimates. Instead the schedule was based on when the capability was desired. In addition, the sponsor and institutional management chose a set of goals that appealed to the funding agency but were not the highest priority for the ultimate customers, the product engineers. The customers needed and wanted a different set of capabilities. They thus had little interest in the initial code project. Once it became clear that the schedule was almost a factor of three too optimistic and that the initial goals were not appropriate, the project goals were changed to match the needs of the customers and a more realistic schedule was developed.

#### 5. Workflows and Tasks

The institution that managed the Falcon project has had decades of experience developing and using similar (but less capable) simulations. However, that experience was in serial development (i.e. develop one capability and test it, then develop a second capability and add it to the first, etc.). Serial code development would have taken 20 years or more to achieve the desired capability. The Falcon code project and others begun at the same time planned to develop the major components in parallel to speed up the overall development process (Figure 4). Component development in parallel placed new and much greater demands on project management issues since the code teams were four to five times larger than in the past. It also called for better risk management techniques. If many components are needed for the full capability, one failure would double the overall development time. This risk was realized for the Falcon project. A contract support group did not deliver a key component. The Falcon team has had to develop it. This has subtracted from the resources

available for other tasks and has delayed realization of the full project capability. The institution has had to learn how to organize and manage this new kind of code development process.

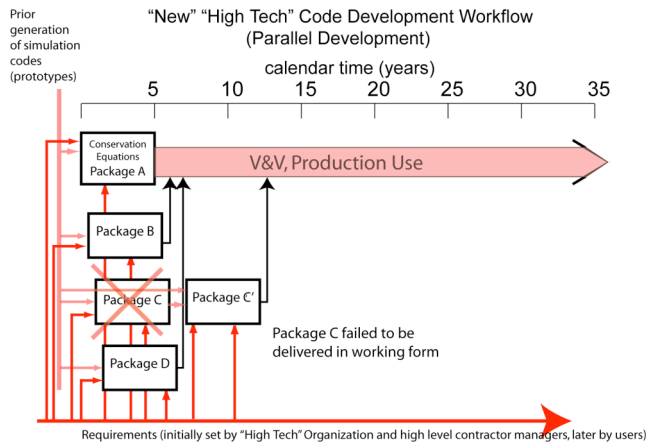


Figure 4. Parallel code development plan.

The tasks that the Falcon code development project and their users carry out can be grouped into seven categories (Figure 5, Table 1).

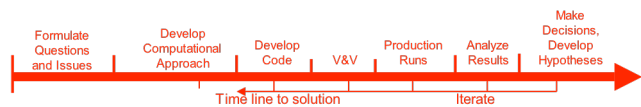


Figure 5. Code development and application task categories.

The tasks are not carried out linearly as in a “waterfall” model. They are nested, and iterative. For instance, a candidate solver might be selected during the design phase. Then it might be discovered during the testing phase or during production runs that it does not provide the needed capability. Then the team has to go back, identify a new candidate solver, develop it, test it, etc., until a satisfactory solver has been found. Or one might discover in the V&V phase that the models miss an important effect that has to be included, and so on. Nonetheless, the use of these categories has been useful for ensuring that all of the tasks are identified for the hardware and software vendors (Table 2). Improved tools to accomplish these tasks would improve the ability of code teams like the Falcon code team to develop scientific codes more quickly with fewer defects and better performance.

The Falcon project also focused on verification and validation. They found that verification and validation has been very challenging. None of the existing techniques for verification have proved to be satisfactory for a complex, multi-physics code. Validation has been similarly challenging. There is little data available, and usually it includes many effects. Identification of the role of specific individual effects has proved to be difficult.

A key observation by the Falcon Team is that debugging massively parallel programs is hard. The worst debugging situations included situations where: the bug is not consistently reproducible, very subtle errors that build from the least significant digit over many calculation cycles, the bug vanishes when the debugger is used, the bug is only reproducible after a very long run (restarting near the bug makes it go away), the bug is not reproducible in a debug version (e.g. you must search in an optimized version where variables have been optimized away and

cannot be viewed), the bug only shows up in a huge problem (giga-bytes of state data) and bugs that occur when the mesh data migrates between processor/nodes (during remaps).

Table 1 Seven Categories of Tasks for Computational Science

1. Formulate Questions and Issues  
Identify high level goals, customers and the general approach
2. Develop Computational and Project Approach  
Define detailed goals and requirements, seek input from customers, select numerical algorithms and programming model, design the project, recruit the team, get the resources, identify the expected computing environment,
3. Develop Code  
Write and debug code, including code modules, input and output, code controllers, etc.
4. Perform V&V  
Define verification tests and methodology, utilize regression test suites, define unit tests and execute them, define useful validation experiments, design validation experiments, get validation results and compare with code results, etc.
5. Make production runs  
Setup problems, schedule runs, execute runs, store results
6. Analyze computational results  
Begin analysis during run to optimize run, store and visualize/analyze results, document results, Develop hypotheses, test hypotheses with further runs
7. Make decisions  
Make decisions based on results, document and justify decisions, develop plan to reduce uncertainties and resolve open questions, identify further questions and issues

## 6. “Lessons Learned”

The “lessons learned” include knowledge of the characteristics of a working high performance computing application code project that will allow hardware and software vendors to develop computers and tools that can improve the ability to develop and utilize the applications, and the steps and procedures that the institution and code team can follow to improve their products and time to solution. Some specific Falcon-related opportunities for improvement are listed in Table 2. It is also worthwhile to list some explicit “lessons learned” that emerged from the experiences of the Falcon team that pertain to team and institutional dynamics. Many of the lessons are emphasized in the standard software project management literature[8, 9].

The case study identified the importance of a competent, well-led, and cohesive code development team. An appropriate skill mix (computer science, scientists with domain knowledge, computational mathematicians, code librarians and documentation staff, etc.) was crucial. It is difficult to program the current generation of massively parallel computers. The team members must develop trust in each other and their leadership.

Detailed requirements are difficult to develop for scientific codes primarily because they often involve research with its attendant uncertainties. This reality sets scientific code development apart from more conventional deterministic software development projects (where requirement rigor is the norm). The Falcon code effort requires research and development. However, it is crucial to develop *some* requirements because it is important that the schedule and resources be consistent with the requirements. A development schedule that was a factor of three too optimistic nearly killed the Falcon project. Thus, in spite of the difficulties

and [3]challenges, it is important to make estimates for the expected schedule. While sponsors and institutional management should determine the programmatic direction of a project, the people that set most of the requirements must have considerable domain knowledge, e.g. the users and developers. Inadequate domain knowledge by the sponsors and institutional management contributed to the specification of an overly ambitious schedule and requirements that didn't meet the needs of the customers.

Table 2. Opportunities for improved development tools and development environments.

- 
- Problem set-up tools (mesh generation, etc.)
  - Data storage and retrieval, especially over distributed networks
  - Smoother upgrades for operating systems and tools
  - Better and easier to use compilers and parallel programming models for massively parallel computers (now Fortran with MPI)
  - Linkers and loaders with ability to link many languages
  - Better parallel debuggers
  - Performance analysis tools (hardware and software)
  - Better run schedulers
  - Visualization (office, small workroom, theater)
  - Data analysis tools (V&V and analysis of runs)
  - Testing tools (coverage analysis, software quality,..)
  - Production run configuration and problem logs

Customer focus is important. The users determine whether the code is successful or not. If they can use it to solve their problems, they will be strong supporters and an important ingredient in the success of the code. If the code cannot help them, then they will either ignore the code or be vocal detractors.

A stable and mature development environment is important. Stable development tools and platforms allow the code development team to concentrate on the code development tasks.

These "lessons learned" can be summarized in the statement that attention to sound software project management is important.

The Falcon project also found verification and validation difficult because methods for verification were inadequate, and because there was a paucity of useful validation data.

## 7. Summary and Conclusions

In order to characterize a large-scale computational science project, we conducted a detailed case study of the Falcon code project. We found that it was essential to maintain complete anonymity to ensure that the team would allow us access to a full and accurate set of information.

We drew three major specific, three general, and many minor conclusions from this case study. The first major specific point is that the life time for this project is expected to be around 30 years, much longer than smaller computational science projects such as are found in academia, and much longer than most projects in the Information Technology industry. This ensures that the project team is very conservative in their approach and emphasizes minimizing risks. They have avoided using new and untried computer languages, compilers, code development methodologies, libraries, etc., especially those targeted to a single platform.

Performance optimization has been much less important than being able to port the code to successive generations of platforms. The demonstration that that many projects have life cycles that span many machine generations has had an impact on the DARPA HPCS vendors. The vendors now have a better understanding of the need for stability and incremental steps for software development infrastructure and tools. Second, the specification of the workflow steps has been useful for identifying the areas where hardware and software vendors can improve productivity by eliminating bottlenecks and improving programming efficiency. Specifying the development steps has helped them focus on the most productive areas for improvement. Third, as noted in the prior section, this study demonstrated that it is impossible to set down specific detailed requirements for a scientific code project.

The case study illustrated the importance of sound project management, not just by the team but also by the institution and the sponsor. Overly ambitious schedules set by the sponsor and institution almost destroyed the project and nearly deprived the program of a very promising and important new computational tool. Support by knowledgeable senior management is essential for success. Verification and validation are an essential element of the development and application of a computational science project. Yet, the intellectual basis for verification and validation is insufficient at present for projects like Falcon to verify and validate their code at the level necessary for success.

## 8. ACKNOWLEDGMENTS

The authors are grateful to the members of the Falcon code team for their participation and their patience and forbearance and for allowing the computational science community to learn and benefit from their experiences. This research was supported by USDOE contract W-7405-ENG-36.

## 9. REFERENCES

- [1]. Petroski, H., *Design Paradigms: Case Histories of Error and Judgment in Engineering*. 1994, New York: Cambridge University Press. 221.
- [2]. Yin, R.K., *Case Study Research, Design and Methods*. Third ed. Applied Social Research Methods Series, ed. L. Bickman and D. J.rog. Vol. 5. 2003, Thousand Oaks: Sage Publications. 181.
- [3]. Dongarra, J., et al., *Sourcebook of Parallel Computing*. 2003, Amsterdam: Morgan Kaufmann Publishers. 842.
- [4]. Capers-Jones, T., *Estimating Software Costs*. 1998, New York: McGraw-Hill.
- [5]. Paulk, M., *The Capability Maturity Model*. 1994, New York: Addison-Wesley.
- [6]. Glass, R.L., *Software Runaways: Monumental Software Disasters*. 1998, New York: Prentice Hall PTR. 288.
- [7]. Ewusi-Mensah, K., *Software Development Failures: Anatomy of Abandoned Projects*. 2003, Cambridge, Massachusetts: MIT Press. 276.
- [8]. DeMarco, T., *The Deadline*. 1997, New York, New York: Dorset House Publishing. 310.
- [9]. Thomsett, R., *Radical Project Management*. 2002, Upper Saddle River, NJ: Prentice Hall