Title:

Case Study of the Hawk Code Project

Author(s):

Richard Kendall, LANL ; Jeff Carver, Mississippi State U.; Andrew Mark, HPCMPO; Douglass Post; HPCMPO; Susan Squires, Sun Microsystems, Inc.; Dolores Shaffer, S&T Associates, Inc.

Case Study of the Hawk Code Project

Richard Kendall, Los Alamos National Laboratory, rpk@lanl.gov
Jeff Carver, Mississippi State U.,carver@cse.msstate.edu
Andrew Mark, DoD HPCMPO,amark@hpcmo.hpc.mil
Douglass Post, DoD HPCMPO,post@hpcmo.hpc.mil
Susan Squires, Sun Microsystems, Inc.,Susan.Squires@Sun.com
Dolores Shaffer, S&T Associates, Inc.,dshaffer@stassociates.com

Abstract

The DARPA High Productivity Computing Systems (HPCS) program is sponsoring a series of case studies to identify the life cycles, workflows and technical challenges of scientific code development representative of the program's participants. A secondary goal is to characterize how software development tools are used and what enhancements would increase the productivity of scientific code developers. The studies also seek to identify "lessons learned" that can be transferred to the general computational science community to improve the code development process.

The Hawk code study is the second code project to be analyzed. This project is based at a large institution under the sponsorship of a federal sponsor. The code development team consisted of a computer scientist and two engineers who developed an engineering code which models a manufacturing process.

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

Through the sponsorship of the DARPA High Productivity Computing Systems (HPCS) program, the present authors are performing studies of high-performance computing application code development projects in order to identify their critical success factors. These studies are also intended to help hardware and software vendors identify issues that must be addressed to improve the productivity of the code development process and to develop a body of case studies for the computational science and engineering community.

It is important in studies of this type to maintain the anonymity of the code project, the host institution and the sponsoring agency or company. As a consequence, “Hawk” is a pseudonym and details that might reveal the identity of the code project have been omitted.

This study followed the methodology described in the Falcon case study:

a. Identify the project and sponsors
b. Negotiate case study with team and sponsors
c. Complete pre-interview questionnaire process
d. Analyze the questionnaire and plan on-site interviews
e. Conduct on-site interview with the team
f. Analyze the on-site interview and integrate with questionnaire
g. Conduct follow-up to resolve unanswered questions
h. Write a report and iterate with code team and sponsor
i. Publish the report

2. Code Characteristics

The purpose of the Hawk code development project was to develop a computational predictive capability to analyze the manufacture of a family of composite material products. This would allow the sponsor to minimize the use of time-consuming, expensive prototypes to ensure the efficient fabrication of the products.

The manufacturing problem that the Hawk code addresses is governed by three physical processes:
a. chemical reactions
b. heat transfer
c. fluid flow through a porous medium

The model for this process involves four independent variables: pressure, temperature, void fraction and degree of completion of chemical reaction. The fluid flow and thermal transport occur on different time scales. In the previous generation of the Hawk code, these variables were computed explicitly. The desire to model the manufacturing process in three dimensions and the prohibitive length of time to perform them with the earlier, explicit code—even on supercomputers—resulted in the pursuit of an implicit treatment of the model. The general computational approach is to use operator splitting for fluid flow, heat transport, and the chemical reactions. The general objective of a suite of runs of the Hawk code is to determine the shortest time to manufacture the product with uniform properties.

Hawk employs an unstructured, fixed finite element mesh to represent and resolve the objects to be manufactured. These can exhibit very complex geometries which may require a significant effort to represent in Hawk (months of staff-time). The development of the current version of Hawk began in 1999-2000. The earlier version was originally targeted for a Connection Machine system, with a *data parallel*[1] code architecture, but evolved into a message passing architecture based on MPI (and targeted to machines like the SGI Origin 3900). The future direction, in so far as port to future hardware is concerned, of the Hawk code has been set by the adoption of MPI. Hawk has been successfully ported to hardware developed by SGI (Origin® 3900), Linux Networx (Evolocity® Cluster), IBM (P-Series® 690 SP) and Intel-based Windows platforms. Domain decomposition is employed to promote parallelism and is implemented with Metis©.

Like the Falcon[2] code, Hawk was developed with multiple languages. There are approximately 134,000 lines of executable code in the program library of which 67% are written in C++, and 18% in C. The remaining 15% are in Fortran 90 and Python, primarily. All of the finite element "objects" and object manipulation is coded in C++ (a strong contrast to the Falcon project, which is based primarily on an object-oriented instantiation of Fortran 77); the Fortran 90 code derives primarily from third-party suppliers.

The code has been deployed to internal and external product engineers, who use it with the help of the Hawk development team. Development team support is important even in the production stage because of the challenges associated with gridding the manufactured objects. The number of customers is relative small (tens). In some cases they come from industries external to the sponsor. The small user base for Hawk represents a problem for its long-term survival—the sponsor responds to user demand **(b)**.

3. Code Project and Team

The Hawk Project had two phases. This report focuses primarily on phase 2, which started in 1999-2000. During the second phase the Hawk development project had a formal project development plan (contract) from the beginning. The deliverables and performance expectations were established in this contract with its sponsor. Care was taken to ensure that the resources and schedules were consistent with the objectives. The contract was reviewed annually. The plan captured details like the maximum expected divergence between scalar and parallel runs and the expected conformance to established experimental results---both unusual for scientific codes. On the other hand, the plan was not so rigid that it did not allow for any "surprises," the unexpected that is typical for scientific codes that extend modeling capabilities into new realms (in this case, 3-D). Both the Hawk sponsor and the development team consider the approach consisting of guiding an "agile[3]" team (for example, one that emphasizes individuals and interactions over processes and tools) with a flexible "contract" to be a critical success factor for this project **(a)**.

During both phases of the Hawk project, staffing was approximately 3 FTEs**.** In phase 2, the Hawk development team consisted of three professionals: a computer scientist and two mechanical engineers. The Hawk team believes that having a multi-disciplinary team contributed to the success of the project **(h)**. As one team member stressed:

> "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 it <the code> perform better."

One member of the team covered the engineering and algorithmic aspects of the project. The computer scientist provided special expertise in "compiler optimization and parallel programming." Both of these team members had participated in the first phase of the project (pre-2000). The third member of the most recent phase of the project (phase 2) took primary responsibility for porting earlier code from F90 to C++. The computer scientist served as the team leader. The small size of the team limited the degree of formality required to manage the Hawk development project. Moreover, the team leader was able to play a role in the development of the code, not just in the management of the project. During phase 1, there was also a mathematician and an HPC programmer assigned to the effort. The cohesiveness of the current Hawk team and the shared project framework provided by a flexible development plan made it possible for such a small team to meet the expectations of its sponsors. This is in contrast to phase 1, which lacked both team cohesiveness and a development plan and was considered less successful (f). The sponsor estimated that disruptive behavior of some members of the phase 1 team set the project back two years (g). An illuminating comment regarding team cohesiveness was made by the team leader about the mathematician who participated in the first phase of Hawk: “For the longest time it was like he was speaking German and we were speaking French.” The development of the current version of Hawk was guided by the following principles:

- Generic, modular, efficient, simple
- Visually friendly
- Robust
- Standards-compliant
- Based on open source tools to the greatest extent possible
- Library-oriented

The members of the Phase 2 team were also guided by work principles that enhanced their ability to work together well. This effective working structure may have been factors in their ability to arrive to solution more quickly. Ongoing and frequent communication during development saved time. As one team member pointed out,

> “I am at <a different location> right now and still working on code development. Sometimes I will change something, submit something, and I will get a call from <the team leader>. We talk a couple times a day and we are always up to date on changes so that communication is important” (i).

In phase 2 the team developed clear work roles and responsibilities that encouraged collaboration. There was clear agreement during our interview with the team members where several members commented on this point.

> “Only thing I can say is you need a multi-disciplinary team. It <C++> is not a trivial language to deal with.”

> “In these situations you need an equal mixture of subject theory, the actual physics, and technology expertise. . .I was more involved

> in the subject area and the specifics of the process being modeled. Code development was more <in the hands of other team members> in Hawk 2."

Finally, the phase 2 team established expectations (rules) about such things as code version control that helped minimize conflict.

Phase 1 of Hawk, launched during the early '90s, was anchored to a procedural development approach implemented in Fortran 90. There was a lack of modularity and little potential for code reuse. Portability was also an issue. C++ offered a way to deal with the first two of these problems. Moreover, it appeared to offer an especially attractive way to deal with the problem of ensuring correctness of implementation for the manipulation of mesh elements, which is a central activity of the Hawk code. Other early development priorities included efficiency, parallel scalability, maintainability, extensibility and reduced development time. Of the twenty or so possible software development metrics that could have been used to track the development process, the Hawk team chose to employ:

- Lines of code
- Time-to-fix defects
- Test coverage
- Code performance
- Parallel scaling
- Number of users

The Hawk team recognized that performance was likely to be an issue with an object-oriented approach based on C++ (the Falcon project took a very different approach for this reason). The team judged that it had to limit the use of performance robbing features of C++ like inheritance and templates to ensure that the penalty for making this choice of primary language was no greater than 20%. The 90/10 locality rule seems to apply to Hawk, namely that 10% of the code consumes most of the runtime. For Hawk it is the linear solvers that dominate the runtime. The Hawk solvers, from third-party sources, are coded in C or Fortran and are very efficient. This greatly diminishes the penalty that the use of C++ imposes on the rest of Hawk. An approach like this, contrary to urban legend, shows that C++ can be used competitively in a production-level scientific code (c).

Different message passing approaches were considered by the Hawk development team during the early development phase (MPI vs. HPF vs. Open MP). There were portability concerns about Open MP at the time a decision had to be made and it was judged not to be well-suited for use with unstructured grids. The experience with HPF was discouraging—the learning curve seemed "as steep as with C++ without the benefits."

Hawk was envisioned as a code that would span generations of hardware. Uncertain future technology is a risk that codes with this ambition must manage. The Hawk team made a conscious decision to manage this risk by (1) requiring access to source code from third-party sources, (2) avoiding proprietary solutions (that is, choosing open source equivalents where possible), and (3) developing multiple options as a fall-back position. The hardest risk to manage was that experienced with some open source codes that did not adhere to commonly accepted software standards (e).

4. Code Life Cycle and Workflow

Unlike the Falcon project reported on earlier, the Hawk project did not establish firm expectations about the duration of the life cycle for this code. An earlier incarnation of Hawk was started in the early '90s, so it is fair to say that at least some of the capabilities of Hawk have existed for nearly a decade. The developers of Hawk certainly had the expectation that it would be useful over multiple generations of hardware; however, the current instantiation of Hawk has only existed through one generation.

The development of Hawk has followed the general life cycle described in Figure 1. Hawk has experienced one production release, but there is currently no formal support for this code, due to the lack of support from customers (b). The deliverables described in the software development plan have been completed. Minimal maintenance is performed on a "volunteer" basis. There is a pending request to port Hawk to the Cray X-1.

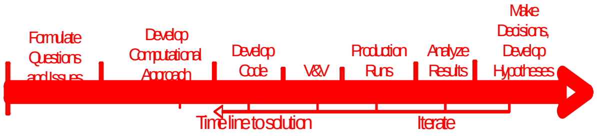
Formulate Questions and Issues
Develop Computational Approach
Develop Code
V&V
Production Runs
Analyze Results
Make Decisions, Develop Hypotheses
Time line to solution
Iterate

Figure 1: The life cycle of a typical scientific code

If we look at Hawk as a 2 phase process, we should consider phase 1 as an experimental phase, although it was not intended as such. The code developed in phase 1 was evaluated and it was decided that important changes were needed if the application was to be successful. This broader view reinforces our understanding of a typical scientific code life cycle, exhibited in Figure 1, as iterative, not linear, to include 1) an initial formulation of solution, 2) code experimentation (intended, or not) and 3) an evaluation stage. The problem was then reformulated to address issues raised in the code evaluation. The middle steps in Figure 1 were repeated in the two phases of the Hawk project.

The development approach for the second phase has been characterized by the Hawk team itself as "iterative" with alpha and beta releases and prototypes. This did not happen in the first phase. Development is done in small increments that can easily be rolled back if a problem is discovered. Nominally, 25% of the project resources were devoted to analysis and design (the first two steps in Figure 1), 40% to code development and debugging, 20% to testing, and 15% to production release and maintenance. These figures were based on the recollections of Hawk team members, not on actual measurements during the development cycle. The Hawk workflow was consistent with the diagram presented in Figure 2.

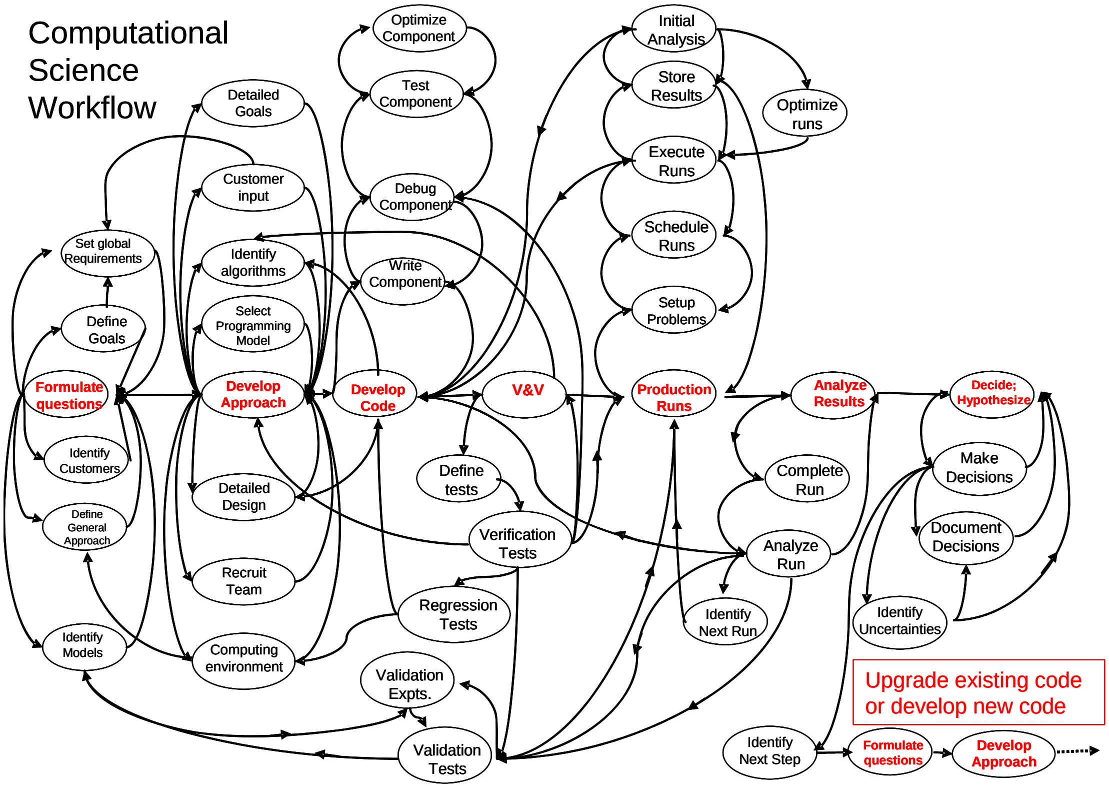

Figure 2: Typical Scientific/Technical Code Development Workflow.

Most of the experience with the Hawk code to date is confined to the left half of the diagram above.

The Hawk development team made significant use of external life cycle management tools. Their choices are summarized in Table 1.

Code Development Environment	
Compilers/Interpreters	C++,C,Fortran, Java
Scripts	Python
Debuggers	Valgrind, gbd
Performance Monitoring	Speedshop, PAPI
Domain Decomposition	Metis
Execution Environment	
Element Generation	CAD ProE
Visualization	ICE, VTK, Paraview, Tecplot
Data Analysis	XDMF (supports Paraview)
Code Development Process Tools	
Configuration Management	CVS
Bug Tracking	Custom (~Bugzilla)
Code Documentation	Doxygen
Support Libraries	
Computational Mathematics	PETc,VSS,PSPASES,CG
Parallel Programming Libraries	MPI

Table 1: Hawk Life Cycle Management Tools

The Hawk code verification procedure emphasized two approaches:

- Comparing code results to a relevant problem with an exact answer
- Comparing calculated with expected results for a problem specifically manufactured to test the code (the so-called method of manufactured solutions[4])

The Hawk development team also relied on the preservation of conserved quantities and the preservation of symmetries as indicators of the correctness of the code.

Validation focused on

- Controlled experiments conducted in the past
- Recent controlled experiments designed to certify the performance of a component of the system

The Hawk development team has developed a test suite that exercises 51-75% of the code, which is typical of such projects. Atypical is the fact that the Hawk program development plan established a priori bounds on the acceptable conformance between scalar and parallel runs of the code (within 1-2%) and the conformance between simulations and experimental data (within 32%) from a history of past performance and insights gained from the first phase of Hawk.

The Hawk project was managed to CMM level 2 (CMM2)[5]. By this we mean that CMM2 attributes (such as documentation, a software development plan, measures of effectiveness, etc.) were required by the sponsor; however no formal accreditation by SEI was ever sought.

In summation, the Hawk code is at a cross-road. It is a good example of a well-managed scientific code development project that has not identified enough customer support to ensure its long-term viability **(b)**.

5. “Lessons Learned”

The Hawk study has reinforced some lessons learned in previous studies:

a. Successful scientific code development in small projects tends to be “agile”, but planning <u>is</u> important to the success of scientific software development projects.
b. Customers, not marketing departments or even sponsors, determine the long-term fate of these codes.
c. Higher level languages (e.g. C++) can be successfully deployed in the high performance milieu if used with care.
d. Portability is essential for long life cycle codes.
e. Risk management is important to the success of long-term technical software development projects. An emerging risk is the dependence on externally-developed tools, but the Hawk team considered the failure to adhere to standards of open source tools (ANSI, best programming practices) to pose the greater risk.
f. Small code teams with only two or three members can operate successfully with a minimum of processes if the team is highly skilled and there is sufficient planning and interaction among the team members *(this is a corollary to a.).*
g. Management support to limit damage by disruptive or uncooperative team members is essential to success.

h. A multi-disciplinary team can be very efficient and effective. In fact, the Hawk team believes that success was achieved because of a multi-disciplinary team approach that combined expertise in engineering, mathematics, programming and optimization.
i. During the formative stages of code development, good communication is critical. The Hawk team evolved from one that was initially co-located to one that is widely geographically separated. The well-established patterns of communication developed early on made this possible.

6. References

1. Dongarra, et. Al. Sourcebook of Parallel Computing, Elsevier Science, San Francisco (2003), page 50.
2. Post, D.E, Kendall, R.P. and E.M. Whitney, “Case Study of the Falcon Code Project,” 2nd Workshop on HPC Applications, ACM/IEEE International Conference on Software Engineering, St. Louis MO, May 22, 2005.
3. see http://www.agilealliance.com/
4. Roache, P.J. (2002), “Code Verification by the Method of Manufactured Solutions,” Trans. ASME 124, pages 4-10.
5. see http://www2.umassd.edu/SWPI/processframework/cmm/cmm.html