



**Fermi National Accelerator Laboratory**

**FERMILAB-Conf-95/065**

## **Software Engineering Methods and Standards Used in the Sloan Digital Sky Survey**

**D. Petravick, E. Berman, V. Gurbani, T. Nicinski, R. Pordes,  
R. Rechenmacher and G. Sergey**

*Fermi National Accelerator Laboratory  
P.O. Box 500, Batavia, Illinois 60510*

**R.H. Lupton**

*Princeton University Observatory  
Princeton, New Jersey 08544*

**April 1995**

*Presented at the Fourth International Workshop on Software Engineering and Artificial Intelligence for High Energy and Nuclear Physics, Pisa, Italy, April 3-8, 1995*

## **Disclaimer**

*This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.*

# SOFTWARE ENGINEERING METHODS AND STANDARDS USED IN THE SLOAN DIGITAL SKY SURVEY \*

DON PETRAVICK, EILEEN BERMAN, VIJAY GURBANI,  
TOM NICINSKI, RUTH PORDES, RON RECHENMACHER, GARY SERGEY  
*Computing Division, Fermi National Accelerator Laboratory, P.O. Box 500  
Batavia, Illinois 60510, United States of America*

and

ROBERT H. LUPTON  
*Astrophysics Department, Princeton University Observatory, Peyton Hall, Ivy Lane,  
Princeton, New Jersey 08544, United States of America*

Received (April 3, 1995)  
Revised (April 3, 1995)

We present an integrated science software development environment, code maintenance and support system for the Sloan Digital Sky Survey (SDSS) now being actively used throughout the collaboration.

## 1. Introduction

The Sloan Digital Sky Survey<sup>1</sup> is a collaboration between the Fermi National Accelerator Laboratory, the Institute for Advanced Study, The Japan Promotion Group, Johns Hopkins University, Princeton University, The United States Naval Observatory, the University of Chicago, and the University of Washington. The SDSS will produce a five-color imaging survey of 1/4 of the sky about the north galactic cap and image  $10^8$  Stars,  $10^8$  galaxies, and  $10^5$  Quasars. Spectra will be obtained for  $10^6$  galaxies and  $10^5$  Quasars as well. The survey will utilize a dedicated 2.5 meter telescope at the Apache Point Observatory in New Mexico. Its imaging camera will hold 54 Charge-Coupled Devices (CCDS). The SDSS will take five years to complete, acquiring well over 12 TB of data.

Fermilab, as part of its Experimental Astrophysics program, is participating in the SDSS, and has major responsibilities for the project's software and software engineering practices.

---

\*Sponsored by DOE Contract number DE-AC02-76CH03000 and the Alfred P. Sloan Foundation.

## 2 *Software Engineering Methods and Standards*

The goals for the software were:

- Allow scientists of all the participating institutions to contribute software.
- Allow institutions to manage large software projects substantially independently.
- Allow very efficient image processing code.
- Minimize costs by re-using software where feasible.
- Identify all software used to produce the survey's data products (from executables to source).
- Produce a system portable to POSIX-like systems.

In contrast with current high energy physics at Fermilab, SDSS had no history, legacy software and development practices with which to contend. We had the opportunity from the start to develop and impose software methods and standards appropriate to distributed development across six sites (including Japan). Our emphasis is on a software engineering methodology that pays much attention to the support and integration of multiple programming environments.

Through two years of development and support we have successfully organized the development of greater than 200,000 lines of source, and integration of more than one million lines before the start of data taking. We continue to improve our basic model and add tools in support of the methods. We can now place more emphasis on developing tools for quality control and analysis with the knowledge that they can be applied across the collaboration software.

### 2. **Baseline**

In 1992, the survey decided to follow the spirit of the POSIX standards, and to incorporate them to the greatest extent feasible. ANSI-C, Extended F77 and C++ were chosen as allowed programming languages.

The survey decided to rely on freely available software, providing three criteria were met:

- The software was reasonably consistent with a POSIX environment.
- The source code was available and could be managed by the survey.
- The survey could imagine maintenance and development of this type of code.

Other technologies were chosen as well:

- X11, with prejudice against proprietary layers, like Motif.
- Use of the World-Wide Web<sup>2</sup> and HTML for distributed documents.
- T<sub>E</sub>X / L<sup>A</sup>T<sub>E</sub>X for document formatting.
- CVS<sup>3</sup> and RCVS<sup>4</sup> for source code distribution and code change maintenance.
- Fermilab UPS<sup>5</sup> software for management of compiled software.
- tcl<sup>6</sup> as an interpreter (followed later by Tk as a GUI).
- Postscript.

- Use of an Object-Oriented Database for the survey's DBMS.
- perl as a scripting language.

### 3. Source Code Methods

The bulk of code written for the survey is in ANSI C. Most computing professionals are reasonably fluent in some variant of the C language, but many scientists were not fluent programmers, and many senior scientists had legacy code in FORTRAN, not C. A major problem for the engineering tools was that people from all levels of skill have to produce portable, maintainable code.

We decided to address this problem governed by the following principles:

- (i) We would insist on strict ANSI C for science code, and insist on using language features like prototypes, which increase compile-time checking.
- (ii) Computing professionals would supply framework code that interfaced to the exterior environment, and so would deal with byte-swapping, and other interface issues.
- (iii) Science code must compile without warnings.
- (iv) We would check for other, more subtle rules via Quality Assurance audits.

From these principles, we constructed just three tools which have provided effective control of these problems.

The first level of control is provided by a makefile driver, `sdssmake`, which sets our standard, picky compiler flags and analyses the standard error output of `make`. It is a survey standard that all compilations use `sdssmake`, and that no warning lines are output to the terminal. `Sdssmake` sets picky ANSI, optimization and debugging flags for all compilers supported by the survey. In particular, we set flags insisting on ANSI prototypes, and the maximal amount of checking. The standard error from `make` is passed through a perl script, `filter_warn`, which suppresses the specific warnings from each compiler which have been determined to be spurious.

The second level of control is provided by a checking script, `quasar`, which is combined with a source code quality audit for each minor milestone. `Quasar` is a perl script which analyses code and object archives, looking for violation of our more obscure survey coding standards. Examples of the kind of checking that goes on are:

- Flag classic C includes, not supported in ANSI C (e.g `malloc.h`)
- Inspect object files using `nm` to see if all defined names obey name discipline standards. For example, no one has defined a routine named "select".
- Check C source for absence of conditional code, excepting conditionals on `NDEBUG` and use of the preprocessor to comment out large blocks of code.

A third level of control is just now being implemented. A script called `audit` is used to inventory just what programs and files have been used to build a product.

Audit is based on the /proc file system. Using /proc, one can trace the execution of a process, tracking the system calls. It is possible to track calls to fork(), and apply the audit to all processes ultimately invoked by some root process. Using /proc, we can inventory the complete context for every exec as well -

- The file executed
- The switches it was passed
- The processes' environmental variables
- Any shared libraries the process mapped to
- Which files the program read

When we apply audit to make, we can establish that

- Compilations were done with the proper flags
- Only supported executables were used to process the source code.
- Only controlled versions of headers files were used during compilation.
- Only controlled versions of object files were used by ld.

#### 4. Distributed Development Environment

Like high energy physics experiments, the SDSS depends on scientists at participating institutions to deliver a considerable amount of software development effort. The SDSS software is developed at collaborating institutions, at the observatory, and by single participants at miscellaneous locations.

The requirements on the distributed development system are:

- Any collaborator can see the source to any of the survey's code.
- Write access to source be controlled by its coordinator.
- Remote developers have a standard survey environment.
- Documentation be made widely available.
- The development system be essentially free.
- The development system be quick to install.
- The development system not rely on the presence of a fast network.

The key component of the distributed development system is the CVS/RCVS code library. The survey has three distinct master libraries, two at Fermilab and one at Princeton. These libraries manage well over one million lines of source code.

Unlike most source code management schemes, CVS is based on optimistic locking and automatic merging of differences. The CVS system permits more than one person to work on a source code file at a time. Because the project maintains an organization chart identifying a coordinator for each software module unplanned collisions do not happen, and planned collisions usually involve minor fixes and maintenance. In this case the merging tools work well.

RCVS's use of rdist allows us to furnish access to the survey's code repository at no cost to collaborators. We have found that our collaborators would incur

unacceptable costs and resist our system if we had chosen to use a file system like AFS.

We can report that 61 distinct computers and 91 distinct users are registered to use our source code libraries. The optimistic locking of CVS causes no significant problems, and the system fundamentally works.

We decided to document the system using the World Wide Web (WWW). The survey was one of the first users of WWW. It has relied on outside software for browsers and infrastructure, like server frameworks. Much of the survey documentation is written in HTML, rather than being converted from another format. Because it was an early user of HTML, and because HTML is the primary language for much of our documentation, we developed, and continue to use several custom tools.

The perl script `html2latex` converts HTML files to latex, allowing us to build hard-copy versions of much of our documentation. The perl script `c2html` converts our template headers in our C language source code to HTML. It is used for automated generation of reference documentation.

The Fermilab UPS system is used to manage the survey's binaries. We have made extensive use of its support for multiple versions of software installed on one computer. We have extended it with our own distribution machinery, supplying our collaborators with releases of all of our underpinnings. This tool, UPR<sup>7</sup> is based on the `rdist` utility, like RCVS.

## **5. Retained Data and Data Model, Object Oriented DBMS**

In 1993, the survey decided to use an object-oriented DBMS for its persistent data. This introduces the C++ language into the survey. We report mixed results, both in the use of the database, and the acceptance of the C++ language.

At a high level, we can report the following:

- We have found object oriented databases to have good speed, and have found that the pain of porting our applications from one of these system to another is less than expected.
- C-Front based C++ compilers are very slow, compared to equivalent ANSI C compilers.
- In 1993, C++ compilers had bugs causing unnecessary conditional compilation in otherwise vanilla source code. For example, one compiler would lack a signed character type, another lacked the `offsetof` operator.
- Object oriented databases, in addition to being licensed software, require proprietary compilers, making it expensive for collaborators to develop software.
- There are people who are productive in FORTRAN and C who are not productive in C++. Use of C++ excludes people who would otherwise have a satisfactory development role in the survey's software. It is the human resources that count.
- Given a decision to use an object oriented database, one still needs to cope

with the interchange of data outside of the project. This is more true in astrophysics than for high energy physics. In astrophysics, these file formats are tabular, not object oriented. Therefore, we have encountered a state-of-the-art problem – flattening our object oriented data model into tabular forms.

Because of these difficulties, science sensitive code is in ANSI C , with C-like data abstractions – structs, not classes. We interface these to C++, and use C++ almost solely in conjunction with the object oriented databases, where the speed and use of persistent pointers are seen as very advantageous.

Because we are in a mixed language environment, our data model must control not only the C++ classes in the object oriented databases, but also C structs and the format details of flat files in interchange format.

## 6. Summary

The SDSS was one of the first major experiments at Fermilab to use UNIX, C and C++. The Survey has developed a distributed, UNIX-based development system at minimal cost by re-using mostly free software tools. The system has been functioning satisfactorily for two years, and supports more than 1,000,000 lines of source. We continue to evaluate its effectiveness and to extend it where necessary. The system plays an important role in developing the skills of all who contribute software to the survey. We continue to be enthusiastic about object-oriented databases, though C++ has failed to be routinely used in science code.

## References

1. "A Digital Sky Survey of the Northern Galactic Cap", Proposal, December 20, 1990.
2. T. J. Berners-Lee, R. Cailliau, J. Groff, and B. Pollermann, "World-Wide Web: The Information Universe," *Electronic Networking: Research, Applications and Policy*, Vol. 2 No. 1, Spring 1992, (Meckler Publishing, Westport, CT, USA), pp. 52-58.
3. Per Cederqvist, "Version Management with CVS," unpublished manual, Signum Support AB, March 1993.
4. Terry Hung, "RCVS: Remote extension of concurrent Versions System," unpublished document, Stanford Linear Accelerator Center.
5. Fermilab Computing Division, "UPS User's Guide," unpublished manual, PN-426, Fermilab, March 1993.
6. J. Ousterhout, et. al., "Tcl: an Embeddable Command Language," *Proceedings of the Winter 1990 USENIX Conference*.
7. V. Gurbani, "Product Distribution Via UPR," unpublished manual, Fermilab, December 1993.