

Code Sharing and Collaboration: Experiences from the Scientist's Expert Assistant Project and their relevance to the Virtual Observatory

Anuradha Koratkar^a, Sandy Grosvenor^b, Jeremy Jones^c, Connie Li^d, Jennifer Mackey^d, Ken Neher^a,
and Karl Wolf^d

Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218

^bBooz-Allen Hamilton, 7404 Executive Place, Seabrook, MD 20706

^cNASA Goddard Space Flight Center, Greenbelt, MD 20771

^dCommerce One, 1100 West Street, Laurel, MD 20707

ABSTRACT

In the Virtual Observatory (VO), software tools will perform the functions that have traditionally been performed by physical observatories and their instruments. These tools will not be adjuncts to VO functionality but will make up the very core of the VO. Consequently, the tradition of observatory and system independent tools serving a small user base is not valid for the VO. For the VO to succeed, we must improve software collaboration and code sharing between projects and groups.

A significant goal of the Scientist's Expert Assistant (SEA) project has been promoting effective collaboration and code sharing among groups. During the past three years, the SEA project has been developing prototypes for new observation planning software tools and strategies. Initially funded by the Next Generation Space Telescope, parts of the SEA code have since been adopted by the Space Telescope Science Institute. SEA has also supplied code for the SIRTf planning tools, and the JSky Open Source Java library.

The potential benefits of sharing code are clear. The recipient gains functionality for considerably less cost. The provider gains additional developers working with their code. If enough users groups adopt a set of common code and tools, de facto standards can emerge (as demonstrated by the success of the FITS standard).

Code sharing also raises a number of challenges related to the management of the code. In this talk, we will review our experiences with SEA – both successes and failures, and offer some lessons learned that might promote further successes in collaboration and re-use.

KEYWORDS : collaboration, code re-use, open source, virtual observatory

1. INTRODUCTION

In the traditional observatory/mission, hardware has been of prime importance and software has provided the tools to manipulate the hardware or the data obtained by the hardware. In this context the software tools were often highly system dependent and served a small user base. But in the era of the Virtual Observatory (VO), software tools will be the primary “instruments” and will perform the functions that have been traditionally performed by physical observatories and their instruments. These tools will not be adjuncts to VO functionality but will make up the very core, the “facility instruments”, of the VO. Consequently, the tradition of observatory and system dependent tools serving a small user base is not valid for the VO. The VO software tools must be designed and developed with a wider audience in mind. Developing a facility class observing instrument is not an easy or cheap task. Neither is developing good software. Furthermore, the economic realities of doing science indicate that we must make software both more capable and less expensive. Hence, for the VO to succeed, we must improve software collaboration and code sharing between projects and groups.

Our working definition of code sharing defines it as code that is developed by one group and made available to other groups as free or lightly restricted software. Sometimes only executable libraries are shared. Other times the source code is also made available. Often the software is made available with no other commitment for future development or maintenance. Shared code is usually very inexpensive for the users (not necessarily for the developers). It is pre-tested and it is often immediately available. New software can layer on the shared code to provide additional functionality. Software reuse¹ refers to taking components of one product in order to facilitate the development of a different product with different functionality. In this context, software reuse is a form of code sharing.

Collaboration includes code sharing but expands upon the concept by including continued parallel development by two or more groups. As its name implies, it is a "collaborative" effort. Changes to the main body of code need to be coordinated among the groups of software developers involved. Each group may proceed to add new code or change existing code. At various points in time, changes from other groups are integrated back into the main code base. The result is a software system that is usually more flexible and adaptive to changing requirements. The ability to adapt becomes inherent in the collaborative integration effort.

2. BENEFITS TO COLLABORATION AND CODE SHARING

The potential benefits of sharing code and collaboration are clear. The recipient gains functionality for considerably less cost. The provider gains additional developers working with their code. The software development process is shortened, and the time and cost of maintaining a product is reduced. In fact, software reuse has a greater impact on maintenance than on development. Since costs of maintenance to development are typically 2 to 1, reuse can play an even greater role in reducing the total cost over a product lifetime³. For example, Hewlett-Packard implemented a reuse program for the San Diego Technical Graphics (STG) division where productivity increased by 40%. The program cost \$2.6 million but has saved over \$5.6 million. The cost of developing the reusable firmware component was only 11% more than the cost of a similar non-reusable component. Integration costs were 19% of the cost of developing a non-reusable component. Therefore, each time the component was reused, the cost was only about one-fifth of the cost of developing that component from scratch⁴.

3. BARRIERS TO COLLABORATION AND CODE-SHARING

There are a variety of barriers that tend to impede the development and acceptance of software code sharing and collaboration. Programmers often have the "not invented here" syndrome. It is sometimes felt that software not developed within the group/organization is not as good as software developed within the organization. This mind-set is obviously corrosive to effective use of other people's code. The "not invented here" sentiment is also strongly echoed by management and organizations who not only feel that software developed by other groups is not as trustworthy, but also that they have no control on the quality of the software that is being re-used. This sense of mistrust is often held with few if any software engineering metrics, either supporting or disproving.

For code to be shared effectively, it generally has to be created with re-use in mind. This has implications and limitations in how the software is designed and coded, so re-use should be adopted early in the software development life cycle. Developers, especially those working under very tight budgets and schedules, often do not feel they can afford the added infrastructure to properly support code sharing. While documentation is important for all software projects, it is particularly important when the code is to be shared. Complete and detailed application program interface (API) specifications must be negotiated and defined often accompanied by the protests of programmers under pressure.

In addition, there may be legal issues to contend with. For example, how is the code to be licensed? What liability do the owners of the code have? For international collaborations, there may be legal ramifications of exporting executable and/or source code. The "open source model" from suppliers of free software such as the Free Software Foundation is a model for such an effort. However, this open and free availability of software may not be appropriate or possible with some software development endeavors.

Collaboration may also have additional difficulties. The physical separation among the interested parties can hinder collaborative development. A common technique during software development is to have one or more brainstorming

sessions to discuss ideas for development and to discuss options for handling problem areas. Handling meetings of this type where the interested parties are physically far apart tends to make this process more difficult. Technologies such as conference calls, collaborative tools such as Microsoft's NetMeeting are useful but they always seem to fall short of the true interactive and productive nature of face-to-face discussions. Collaboration also involves coordination and the management of changes. This costs all involved groups time and money, which are scarce commodities in the science research business.

But the above difficulties are not insurmountable. They just need to be planned for and properly managed. Project management may have issues with supporting code sharing and collaboration. The additional support imposes increased investment costs in both time and budget. Managers need to understand re-use and buy-in to its concepts. Of particular concern is software maintenance. It is probably the biggest impediment to the adoption of code sharing by the user's management. For example, what happens if something is wrong in the code? Who will fix it? How quickly? What will it cost to fix? Who owns the code? Who decides what changes to incorporate? A key to successful software code sharing and reuse has been full management commitment to the reuse of source code modules. In the case of Hewlett-Packard's reuse program reuse was promoted via a number of incentives. The results of these incentives was that software reuse rose to over 20% saving the company more than 1.5 million dollar in the first two years itself. The overall savings by adopting reuse are expected to be well over 10 million in five years ⁵.

For the last 15 years it has been known in the software industry that on an average, only about 15 percent of any software product serves a truly original purpose. The other 85 percent of the software product could in theory be standardized and then reused in future products ². In astronomy, although missions and observatories may be providing unique capabilities, user support tools and data analysis tools can often be similar, indicating that code sharing and collaboration should be seriously considered. Yet, historical evidence in the field of astronomy shows us that the process has not been a success in the past except for a limited number of cases. Adoption of the FITS format, and adaptations of the Space Telescope Science Institute's (STScI) SPIKE scheduling software by a number of observatories are some of the examples where collaboration and code re-use have been successful

A number of factors have arisen in the last five years that indicate we take a closer look at changing the paradigm in which software for astronomy is developed.

- **Substantial improvements in software developments tools and languages.** Software capabilities such as the wide use of object oriented programming languages (e.g. C++ and Java) have made code sharing/reuse feasible by improving packaging and modularization of the code. By compartmentalizing the code into neat, well-defined modules with clear interfaces, it becomes much easier for subsequent users to integrate the shared code into their new software applications. In addition, with improved multi-platform languages (again an example is Java), software can be written for computing platform but be shared without change on a variety of other platforms. Sun Microsystems' slogan for Java, "write once, run anywhere", is now much closer to reality than in the past.
- **Many missions/observatories are developing new tools** or upgrading their old tools.
- **Software funds are shrinking.** It is increasingly difficult for each individual mission/observatory to invest in software research, usability testing etc. to provide the state-of-the-art tools to their user community.

Often observatories/missions are unable to fund general-purpose capabilities in the software they develop because of the tight software budgets. It has been sighted that there be independent opportunities for teams to propose separate software R&D efforts (Science Data Processing Workshop, <http://that.gsfc.nasa.gov/gss/workshop2000/>). The Scientist's Expert Assistant (SEA) is such an effort, which was initially funded by the Next Generation Space Telescope funds and is now being supported by Applied Information Systems Research and Space Operations Management Office's Technology Development Program. In this paper we share our experiences of code sharing and collaboration in the SEA project. Finally we discuss the relevance of our experiences for the Virtual Observatory.

4. WHAT IS THE SCIENTIST'S EXPERT ASSISTANT?

In order to meet the operational cost objectives for the Next Generation Space Telescope (NGST), a software research and development effort called the Scientist's Expert Assistant (SEA, <http://aaaproduct.gsfc.nasa.gov/SEA>) was started to investigate automated solutions for reducing the time and effort spent by scientists and telescope operations staff in preparing detailed observatory proposals. The SEA has developed innovative tools to enable scientists to develop valid observational proposals more efficiently. The goals and philosophies used in developing SEA have utilized an interactive visual and expert system approach to make the user more self-sufficient and hence minimize manual effort and cost for user support.

SEA has used the Hubble Space Telescope (HST) as a test bed. While HST has provided our baseline, we believed early on that substantial productivity gains could be made by having a modular, multi-observatory system, and have taken care to make the prototype's architecture as modular and as general as possible. SEA source code is freely available. *We also believed that collaboration and an evolution to multi-observatory tools is the avenue that promises the greatest cost benefit.* There are major cost benefits to observatories if they can support a commonly used tool and adapt it with a small effort, rather than redevelop (and subsequently maintain) a fully custom tool. In addition, there are major productivity gains to both support staff and astronomers if different observatories use the same basic software suite of tools (similar to the adoption of FITS format). Lastly, if astronomers do not constantly have to learn new proposal preparation tools, their ability to focus on their science will allow substantial improvements in the efficiency and quality of the observations.

5. SEA CODE SHARING AND COLLABORATION EXPERIENCES

While SEA was initially conceived as a research and development effort for NGST, the NGST observatory was not sufficiently concrete for use as a testbed. The SEA thus used HST as its testbed observatory. However, a goal of the design was to produce something that could scale to support NGST once the NGST design was finalized. As a result, SEA was developed with multiple observatories in mind; hence, we aggressively pursued the transfer of code and collaborations with other observatories and missions. This was achieved by:

1. **Sharing the entire code base** so that it could be adapted and used as an operational system by an observatory. The transfer of code from the SEA to the Space Telescope Science Institute (STScI) was our first full-scale code-sharing challenge. In this case, STScI agreed to produce operational tools for HST using the entire SEA source code. This effort was dubbed the Astronomer's Proposal Tools (APT, <http://apt.stsci.edu/>). The SOFIA observatory also considered adapting SEA code for their operational use, but due to budget difficulties has withdrawn their offer.
2. **Sharing individual modules** with observatories. The SIRTf observatory has reused limited modules of the SEA code.
3. **Collaborating with other groups**. Through the JSky (<http://archive.eso.org/JSky>) effort, the Gemini Observing Tool (OT) and SEA share several modules that are common between both tools. Both groups continue to enhance the common modules. We are also collaborating with the European Southern Observatory's (ESO) Very Large Telescope (VLT) team to testbed a data simulation pipeline in SEA.

5.1 HST Experience

Prior to the transfer of code to STScI, our experiences with sharing code were limited to isolated requests for individual classes with no coordination of changes. The goal of transferring code was to setup a collaborative partnership since our SEA development was to be an ongoing effort. We needed a system that would enable both groups to develop the code, but somehow also remain synchronized.

The initial thought was to follow the model of many popular open source initiatives such as the Apache projects or NetBeans. Those efforts maintain a core CVS repository that is accessible by many authorized parties who may checkout files and make changes individually. We felt this would achieve maximum synchronization while still allowing branching outside of the repository. However, STScI management would not agree to this approach. Their

concern was that they would not have the necessary control over the integration and management of changes, and thus quality control would be an issue. Since the two groups had differing environments, STScI is a production, operational environment, while Goddard's SEA team operates in a more R&D environment, the STScI management was not comfortable with the safeguards involved in a single code base between the two environments. We were thus required to rethink our approach.

The final solution was for both parties to start with the same code base, but maintain their changes in separate, locally controlled repositories. This allows the STScI to have the control they needed over changes to their version of the code. However, both sides still wanted to reap the benefits of shared work on the project, so we decided that we would continue to pass changes each team made on to the other team for integration into their repository. Neither side was obligated to accept changes. However, both understood that rejecting changes increases divergence between repositories, so the ultimate goal was to incorporate changes as much as possible. This simple model is reasonable only because the number of teams was small (two) and the number of developers was small (4-5 per team).

Our experience so far with this model has been fairly positive. At one point, our team made a substantial change to the entire library, which the STScI was able to integrate with few difficulties. In addition, on several occasions, we have sent them enhancements that they integrated into their version, or they provided bug fixes that we incorporated into our version.

A minor stumbling block was user interface (UI) changes. It was reasonable to expect that each side would have different opinions on the look and feel of the UI. For this reason, certain UI modifications were disabled when the changes were incorporated. We decided to do this rather than discard the UI changes to enable easier integration of later changes to the same code. Given that the SEA was a research prototype effort without its own user base, it was acceptable to have different UIs for the SEA and APT. Had there been two distinct user communities, we would have been reluctant to allow deviations in the UIs due to the confusion this would have caused the users.

The code sharing with STScI has enabled the APT group to release their first operational tool to the user community within six months of the start of the APT project and most of the developmental cost was incurred in the SEA research and development effort of the SEA prototype. The APT development budget is primarily allocated for making the prototype operational and expanding the original SEA by integrating more instruments that were not in the original prototype. *We feel that this particular code sharing experience was successful because of the management buy-in, demand from the HST user community, and the fact that SEA was prototyped using HST.*

5.2 SIRTf Experience

Our experience with SIRTf was much more modest. Near the end of our development cycle, SIRTf was beginning a parallel effort to develop tools for their observatory. The SIRTf developers had seen the SEA and noted components that would be useful in their own development. The developers then asked us for specific SEA components, including a world coordinate system grid, NED and SIMBAD clients, and FITS image display components. We complied, and they integrated the components into their SIRTf Planning Observations Tool (SPOT). The process was trivial and essentially consisted of an informal exchange of e-mails. Our collaboration ended there, which was entirely appropriate given that our act of code sharing was, in this case, a one-time event. *This example shows that code sharing can be successful (and is perhaps most likely to succeed) at the small-scale level, where individual developers share specific components or even sections of code. There was no formal or informal management involvement.*

5.3 VLT Experience

The SEA architecture allows for the integration of multiple observatories. Hence, we are in the process of test-bedding the idea by adding the European Southern Observatory's (ESO) Very Large Telescope (VLT) as a new observatory. Our first collaboration effort was to integrate the VLT exposure time calculator with the SEA exposure time calculator. This collaboration between the SEA and VLT teams was done at the grass-roots developer level and was a small effort to bypass the need for formal management budget and subsequent approval. The development effort took place primarily through short intensive collaborations timed to coincide with conferences (and therefore avoid additional travel costs). *Our lesson was that while e-mail could set the stage for a successful collaboration, it takes dedicated time together for*

significant progress to occur. The dedicated sessions do not need to be long; a couple of days may suffice, with ongoing progress continuing via e-mail.

Our second collaboration effort is to integrate the VLT instrument simulation results into the SEA data simulation pipeline. This is once again a developer level collaboration with no formal management involvement. For this effort, the SEA team initially proposed using the Java RMI technology to communicate to the VLT simulator that is running on a web server. This approach will involve a close collaboration and software development efforts on both sides to define an interface from which SEA can communicate with the VLT simulator to post and receive simulation results. Coordination has proven to be difficult due to the different time zones (ESO is near Munich Germany and we are located on the US East Coast), and the fact that this is a small informal research effort for the VLT team and is not a critical part of their operational environment. The solution that the SEA team chose was to enhance and develop a CGI client interface to query the VLT simulator directly. This approach did not require changes or new development effort for the VLT simulator software for the initial work. An implementation is now in place in the SEA Simulation Facility to obtain imaging simulation results for one of the VLT instruments, FORS1.

5.4 JSky Experience

The JSky project (<http://archive.eso.org/JSky>) is an example of a successful collaborative effort. JSky is ESO and Gemini's attempt at coordinating a library of reusable Java components for use in astronomy. The Visual Target Tuner (VTT) module of the SEA uses JSky components for its underlying image rendering engine. This includes a FITS image codec for the Java Advanced Imaging (JAI) API. JSky enabled the VTT to utilize the JAI libraries with minimal effort, increasing the capability and performance of the VTT. The SEA team has also shared code and ideas that have been incorporated back into JSky, such as a world coordinate system library that was originally written for the VTT. *JSky has been a successful collaboration for the SEA and other members of the JSky group because it has actually reduced development time instead of increasing it. Still, JSky is a rather modest library, and we hope that the community will embrace it by contributing additional components.*

5.5 Other Collaboration Experiences

At the 1998 SPIE meeting on "Observatory Operations to Optimize Scientific Return" there was consensus that we explore the mutual concerns and constraints of the various observatories, so that we can understand where joint, shared, or collaborative effort might be beneficial. We therefore organized the "Workshop on Observing Tools" in October 1998 (<http://aaaproduct.gsfc.nasa.gov/workshop/>). Over fifty representatives, a mixture of astronomers and software developers, representing a variety of observatories (ground and space-based, optical, x-ray, infrared) attended this workshop. Our goal in organizing this workshop was to promote collaboration between observing tool development teams. It was our hope that this meeting would begin a dialogue on collaboration and software reuse that would continue after the workshop was concluded.

The meeting went quite well. It was well attended, with representatives from many different organizations including Chandra, ESO, Gemini, HST, NOAO, SAO, and IPAC. Topics ranged from the commonalities between observatory needs, to the future of observing tools, to how to break down barriers to collaboration. While the meeting was declared a success, the group hoped that the meeting would be just the beginning of a larger dialogue and collaboration effort. Six working groups were setup that covered specific observing tool topics. Both short-term and long-term goals were established for each group. However, despite the best efforts of the members, none of the working groups ever produced any substantial results. Some groups started out strong but quickly dissolved out due to lack of interest or time, while others never really started at all.

While it is certainly worthwhile to strive for collaboration between organizations, it is unfortunately extremely difficult to sustain. Clearly the workshop attendees wanted to share their efforts and work towards common solutions. Nevertheless, *in this case collaboration failed because there was no driving force that compelled people to expend the extra effort required to collaborate. Lack of organizational commitment is one reason. Another is the simple reality of daily pressures exerting priority over other "optional" activities. For a collaboration effort to succeed, it must be carefully focused so that the long-term cost of not participating is higher than the initial cost of participating in the effort.*

6. EMERGING DEVELOPMENTS TO PROMOTE REUSE

A new software engineering methodology has recently been gaining much publicity as an effective way to build quality reliable software while increasing the flexibility of software to adapt to the inevitable internal and external design evolutions and revolutions. This methodology, known as eXtreme Programming, or XP, relies on object oriented programming style, small teams, and a rigorous use of unit level testing. While many parts of the XP methodology are controversial, one premise that shows promise for promoting collaboration and re-use is the effective use of extensive unit testing. A basic premise of XP programming is that thorough and repeatable unit level tests are the fundamental building block for code. An open source framework for supporting the development and execution of unit tests is the JUnit suite. Both STScI and SEA teams have independently begun to integrate JUnit tests into their code bases. What this gives is an excellent, reliable, and intuitive way for both teams to execute regression tests to ensure that code changes are performing properly. Since the testing framework is simple to implement and run, JUnit provides not only an effective testing platform, but also an efficient development and debugging tool. This provides effective encouragement for programmers to integrate unit testing throughout their development.

7. RELEVANCE TO THE VIRTUAL OBSERVATORY SOFTWARE DEVELOPMENT EFFORT

As mentioned in the introduction, in the era of the VO software tools will have to be considered as facility class instruments and will have to be developed with a broad range of users in mind. The tools most likely will be developed by individual groups with specific needs to solve for a particular problem. But given the VO operational budget these individual efforts will have to contribute to the global effort by either providing code that can be shared/adapted/extended/reused as common usage software. To achieve this we need to overcome our re-use barriers and change the paradigm in which software is being developed for the astronomy community. From the code sharing and collaboration experiences above we have learned that we need to:

1. Design the software with reuse/portability in mind
 - a. Design for mission/observatory independence where ever possible
 - b. Design for hardware platform independence
 - c. Document all the lessons learned
 - d. Fully document all software interfaces to aid the software developers to integrate their software
2. Establish a knowledge base of reusable software and expertise
3. Consider module/component level reuse
4. Allow software engineers to travel to conferences etc. where they can collaborate on technical issues
5. Consider the Open source process
6. Provide funds for software research and development
7. Have management buy-in to the code sharing concepts. This is essential if we want to achieve code sharing on a large scale.

Not only have the capabilities/complexities of scientific instruments in astronomy increased, but also the amount of data acquired by these instruments has increased exponentially. In this environment, software tools are an essential/vital component to increasing scientific productivity. Given the advances in information technology, there needs to be sufficient research effort (which translates into time and money) in developing state-of-the-art software tools. To make the cost of software research and development viable we will have to develop an environment in which there is efficient software sharing and reuse. *In developing the SEA as a testbed for new visual approaches, we've learned that the really substantial reductions in costs will come not from simply applying technology, but from establishing a common core of tools across all major observatories, ground or space based, inside or outside of NASA.* Having a common core will allow all observatories to save the costs of proprietary development, testing, and maintenance. It will also improve science by saving the end-user astronomers the costs of learning multiple tools, allowing them to gain significant comfort and expertise in a single suite of tools applicable to many observatories. Into the future, a common core will continue to save money as observatories can work together to evolve and improve their software tools, putting software research and development efforts into research, that no single group can afford to do on their own, and thus keep the software tools from stagnating until they become an operations crisis.

ACKNOWLEDGEMENTS

This work was initially supported by the Next Generation Space Telescope and is now funded by the Applied Information Systems Research grant NAG-9512 and SOMO Technology Development Program.

REFERENCES

1. S.R. Schach, *Software Engineering with Java*, Irwin-McGraw Hill, New York, pp.178, 1996.
2. T.C Jones, "Reusability in Programming: A Survey of the State of the Art", *IEEE Transactions on Software Engineering*. **10**, pp. 488-494, 1984.
3. S.R Schach, *Software Engineering with Java*, Irwin-McGraw Hill, New York, pp. 185, 1996.
4. W.C. Lim, "Effects of Reuse on Quality, Productivity and Economics", *IEEE Software*. **11**, pp.23-30, 1994.
5. S.R. Schach, *Software Engineering with Java*, Irwin-McGraw Hill, New York, pp. 184, 1996.