Lessons Learned From Developing and Maintaining Shared Astronomical Software Packages

Jessica Mink[1]

[1]*Smithsonian Astrophysical Observatory, Cambridge, MA, USA;*
jmink@cfa.harvard.edu

Abstract. Software which performs specific useful tasks for astronomers can have a very long life. If the user interface is simple enough and the code is portable, the use can spread quite widely, and if the API's for the included self-contained subroutine library are straightforward, it will be included in other packages and demands on the developer grow. How can one person keep software useful and citable for 30 years, and where does it go from there?

1. Introduction

Over my career, software development media have evolved from cards and tape to bits in cyberspace and email, software conferences, and the World Wide Web have smoothed communication between developers so transfer of code has become easier and easier. Here's what I have found out about sharing software over forty-five years.

2. Planetary Spectra: Software Tied to Science

When I started in the early 1970's, scientists wrote their own software to analyse their own data and maybe that of some close colleagues. Data was on cards or magnetic tape and software was on cards, magnetic tape or paper tape. When I was working on my Master of Science thesis at M.I.T. in 1974 and needed to project data on maps, all I could find in the MIT Libraries was one book with map projection formulae. I turned those into Fortran code to project maps because I could test the Fortran for free on the spare CPU of the computer which I made a living operating at night. When I had to feed data from the PL/I image pipeline into the mapping software, I crashed the IBM 360/65 computer several times before I figured out what was wrong with the interoperation lore which had been passed down to me by word of mouth. In the end it all worked, and magnetic tapes were generated which I could walk to a Calcomp plotter in a different building which drew India ink maps and graphs of reflectance spectra of various portions of Mars. The spatial variation in mineral absorption bands across the planet were obscured by internal reflections in the detector, so no scientific results were published, but I wrote my first data pipeline (Mink 1974). After graduation, I got a job developing and using financial software, learning to compute precisely while also contributing map projection skills in my spare time to my first scientific paper (McCord et al. 1977).

Lessons learned included:

1. Use other people's image processing software if it works
2. It is possible, but not trivial, to write code that works on different computers.
3. Using multiple languages on a project can both save and waste developer time. (It may take trial and error to interface two languages on the same computer)
4. Scientific failures can still teach useful skills.

3. Occultations: Applying Thesis Lessons Successfully

Following my spouse to Cornell, I got a job in lieu of grad school admission developing software to observe and reduce time-series data of occultations of stars by solar system objects. Six months later, our group unexpectedly discovered rings around the planet Uranus (Elliot et al. 1977), my software system grew so that we could monitor the Uranian rings and look for their counterparts around other planets, and our group moved to M.I.T. to replace the group where I did my thesis. I started to learn C because it was easier to read and write metadata and data as strings of bytes in C.

Lessons learned included:

1. Obsessive accuracy is important whether numbers are money or positions on the sky.
2. Results which are testable by observation make for reliable software.
3. Translating software across platforms makes it more robust.
4. Standard data formats such as FITS make data more usable across time

4. Spacelab 2 IRT: Using (or not) Other People's Software Systems

In 1984, I moved across Cambridge to the Harvard-Smithsonian Center for Astrophysics to work on the Spacelab 2 Infrared Telescope, soon to fly on the Space Shuttle and map the infrared sky from 2 to 120 microns. My job was to put together a data pipeline to convert telemetry data to maps of the sky. We planned to use IRAF, a new all-purpose image data processing system, but it turned out to be easier to add to my existing utilities, (Kent et al. 1992). Our workhorse terminals were brand-new VS100 workstations, which had an undeveloped potential to do the black and white graphics our project needed. The X Window System was being developed across Cambridge at MIT so I joined the project (and learned more C) to write the Tektronix terminal emulator to what we needed and at the same time emulate a DEC VT240 text and graphics terminal. My contribution became the first xterm graphics terminal emulator. (X Consortium 1984)

Lessons learned from xterm:

1. Don't buy new hardware unless you are prepared to write system software. (If you need software badly enough, you can build it.)
2. Other people's code with descriptive variable names and lots of comments can be read and used without documentation.
3. Setting development goals is good.
4. It helps to have external tests of your code, no matter how specialized it is.
5. C is portable and powerful.
6. Well-written libraries can be used beyond their original purpose.

Lessons learned about pipelines:

1. Design pipelines while instruments are being designed, not just before launch.

2. Standards are good, but you may need new ones.
3. One system won't work for every project; we should have used AIPS.
4. Inventing a new language raises a huge barrier to developer participation.
5. Persistent parameters are useful for users and I added them to my software.
6. Once you write mapping software, it can be reused for totally different projects
7. Libraries of subroutines with clear APIs and style make that easier.
8. Its easier to read and write arbitrary formats of data with C than with Fortran

5. RVSAO in IRAF: Data Analysis Software on Someone Else's System

As the Spacelab 2 IRT project was closing down, SAO needed new software to analyze data from their high and low dispersion spectrographs on their ground-based telescopes. The facilities already in IRAF needed only a more accurate, pipelinable way to derive radial velocties from large quantities of spectra from our telescopes. A package already existed to cross-correlate spectra, but we needed more options and precision as well as the ability to measure emission line centers to get redshifts. I combined these tasks into the RVSAO package (Kurtz & Mink 1999) and (Mink & Kurtz 1998), which has become a standard across astronomy. We still use it to return analyzed data from five spectrographs to our users the day after the telescope takes it.

Lessons learned:

1. Sometimes, IRAF is very useful.
2. A ubiquitous, maintained software system makes sharing easier.
3. If you use a package every day, most of the bugs get shaken out.
4. If other people use it, too, different bugs get found even faster.
5. Developing useful code in a little-known system means you have to fix all of the bugs.
6. After 30 years the system is no longer as ubiquitous.
7. Write packages in a long-lived, system-independent language.
8. Make packages as self-contained, but universal, as possible.

6. WCSTools: Image Analyis with Standalone Utilities

At the ADASS IV FITS BoF in 1994, Don Wells presented Bill Cotton's worldpos.c, a C library for translating between sky coordinates and image coordinates (Greisen & Cotton 1994). As I was maintaining the SAOImage image display program at the time, it seemed like a good idea to add Bill's open-source package to enable SAOImage's cursor to track sky coordinates in real time, so I wrote a C library to do that, keeping the World Coordinate System code separate from the rest of the SAOImage code (Mink 1996). Few ground-based images had good WCS parameters, so I rewrote my Fortran libraries that read and wrote FITS files and object catalogs to deal with modern images and catalogs in C, upgraded code from the University of Iowa (Downey & Mutel 1996), and created the WCSTools package of utilities and subroutines (Mink 2011).

Lessons Learned:

1. Performing services which everyone needs, but few really understand, sells.
2. A single-language, open-source software package sells.
3. Good web documentation with examples leads to fewer help requests.
4. Policy/GUI-independence helps (Avoid graphics).

5. Top-level subroutines with simple API's also sell.
6. FITS works for now, but needs change over time.
7. Developers don't last forever, so make it easy to turn over responsibility.
8. It helps if package developers have a long-term position and never retire.

7. Preserving Software for the Future

One of the problems with sharing software libraries is that they change continuously, and software which works at one point may not work even a month later. I've dealt with this in WCSTools by keeping the software's subroutines and capabilities limited to the tasks at hand and trying to eliminate any use of system-dependent functionality. Source code for all previous versions of the package have been maintained for 25 years so far (Mink 2019). It is hoped that a Python interface to the entire WCSTools package can be developed over the next several years.

RVSAO is dependent on the IRAF system for data I/O from many formats and graphics support. That makes it susceptible to failure if IRAF is no longer supported, though its 30 years of continual usability are more than most current operating systems have been in use. SAO's spectra data reduction pipelines have used XCSAO and EMSAO for almost 30 years (Mink 2013)! If Python can support access to spectra, the cross-correlation, line-fitting, and equivalent width capabilities of RVSAO will be added.

References

Downey, E. C., & Mutel, R. L. 1996, in Astronomical Data Analysis Software and Systems V, edited by G. H. Jacoby, & J. Barnes, vol. 101 of Astronomical Society of the Pacific Conference Series, 380

Elliot, J. L., Dunham, E., & Mink, D. 1977, Nat, 267, 328

Greisen, E. W., & Cotton, W. 1994, Classic AIPS World Coordinate Systems. URL `http://tdc-www.harvard.edu/software/wcstools/wcstools.aips.html`

Kent, S. M., Mink, D., Fazio, G., Koch, D., Melnick, G., Tardiff, A., & Maxson, C. 1992, ApJS, 78, 403

Kurtz, M. J., & Mink, J. 1999, RVSAO 2.0: Digital Redshifts and Radial Velocities. `9912.003`

McCord, T. B., Huguenin, R. L., Mink, D., & Pieters, C. 1977, Icarus, 31, 25

Mink, D. 1974, Determination of Martian surface reflectivity from 0.4 to 1.1 micron using a vidicon spectrometer. URL `http://hdl.handle.net/1721.1/55007`

Mink, D. J. 1996, in Astronomical Data Analysis Software and Systems V, edited by G. H. Jacoby, & J. Barnes, vol. 101 of Astronomical Society of the Pacific Conference Series, 96

— 2011, WCSTools: Image Astrometry Toolkit. Astrophysics Source Code Library, `1109.015`

Mink, D. J., & Kurtz, M. J. 1998, in Astronomical Data Analysis Software and Systems VII, edited by R. Albrecht, R. N. Hook, & H. A. Bushouse, vol. 145 of Astronomical Society of the Pacific Conference Series, 93

Mink, J. 2013, in Astronomical Data Analysis Software and Systems XXII, edited by D. N. Friedel, vol. 475 of Astronomical Society of the Pacific Conference Series, 291

— 2019, in Astronomical Society of the Pacific Conference Series, edited by P. J. Teuben, M. W. Pound, B. A. Thomas, & E. M. Warner, vol. 523 of Astronomical Society of the Pacific Conference Series, 281

X Consortium 1984, XTERM Man Page. URL `https://www.x.org/archive/X11R6.8.2/doc/xterm.1.html`