Lessons Learned from Developing and Maintaining Shared Astronomical Software

IBM Cards

Lineprinter Output

1974

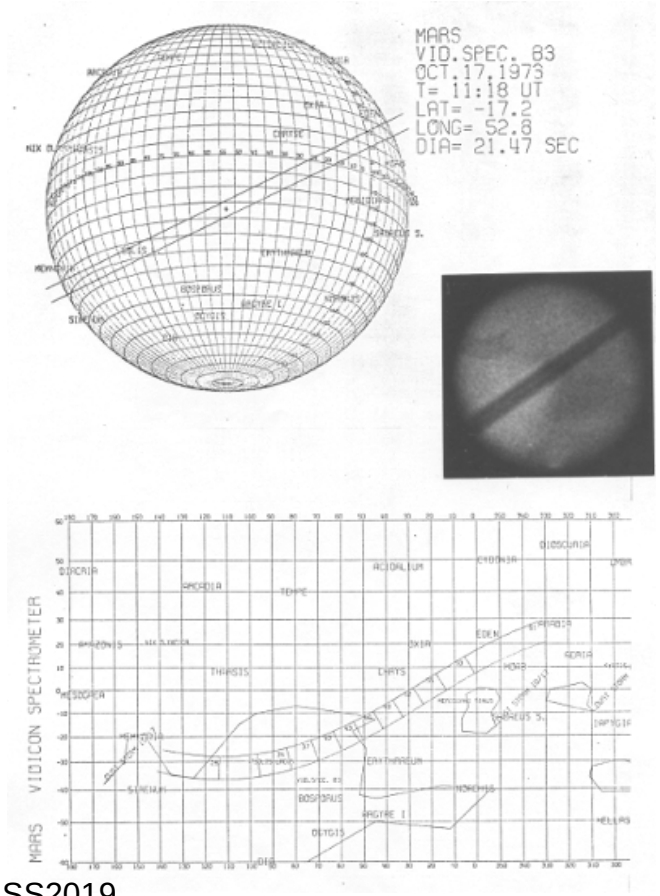
Jessica Mink
Center for Astrophysics | *Harvard & Smithsonian*

2019

45 Years of Using Shared Software

- **Master's Thesis: Develop data reduction system and learn mapping**
- **Financial Publishing: Using software to pinch pennies → computational precision**
- **Occultations: Learn solar system astrometry using JPL software and use FITS**
- **Spacelab 2 IRT: IRAF, xterm, and sharing my own software with more FITS**
- **Radial Velocities: Using IRAF on spectra**
- **World Coordinate Systems:**
- **Maintaining useful 30-year-old software.**

1974 MIT MS: 1973 Mars Opposition MIT Vidicon Spectrometer

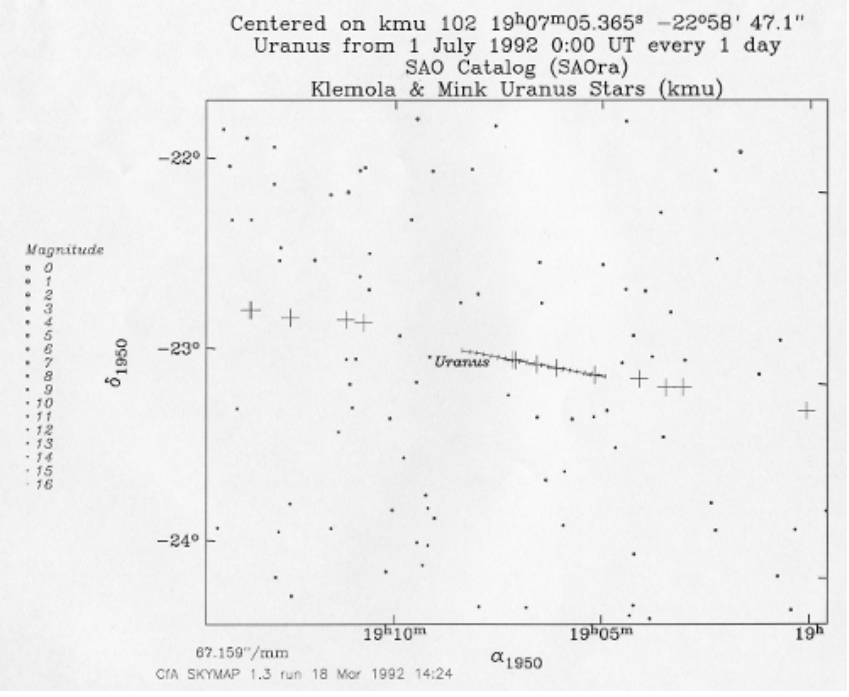


- Slit image projections on film were turned into a projection of the slit across Mars
- Orthographic projection derived from first principles.
- Mercator projection was computed using the only formula I could find in the one book of map projections in MIT's library.

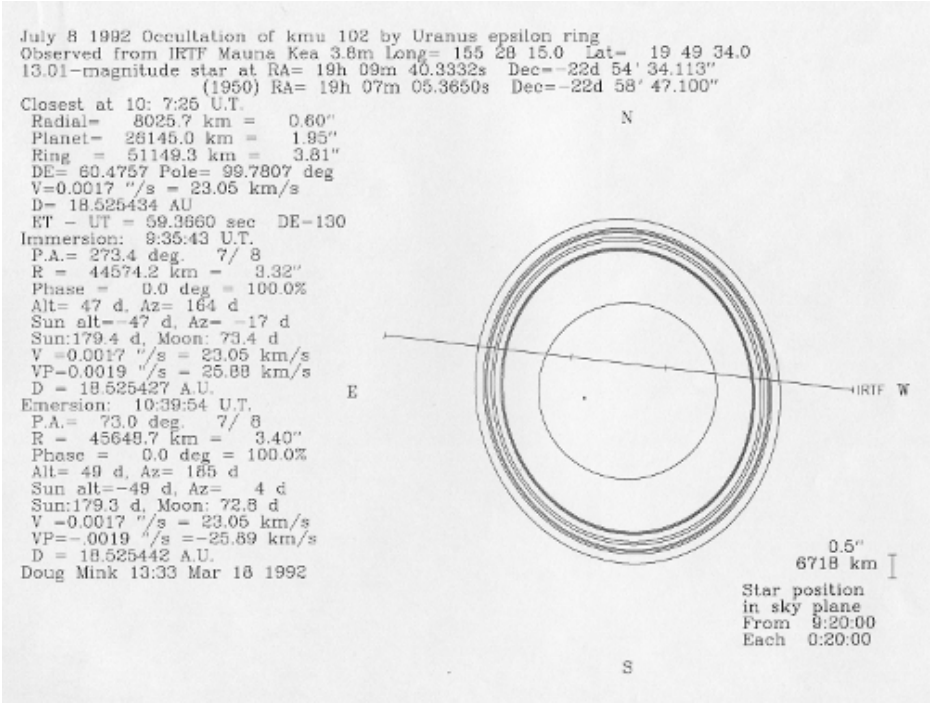
Lessons Learned

- Sharing image processing programs helps
- You can write code that works on both DEC and IBM computers.
- Don't believe lore about interfacing PL/I and Fortran
- Failed science can still teach useful skills.

Occultation Prediction and Data Analysis



SKYMAP output showing Uranus occulting star



OCPRED map in Uranocentric frame

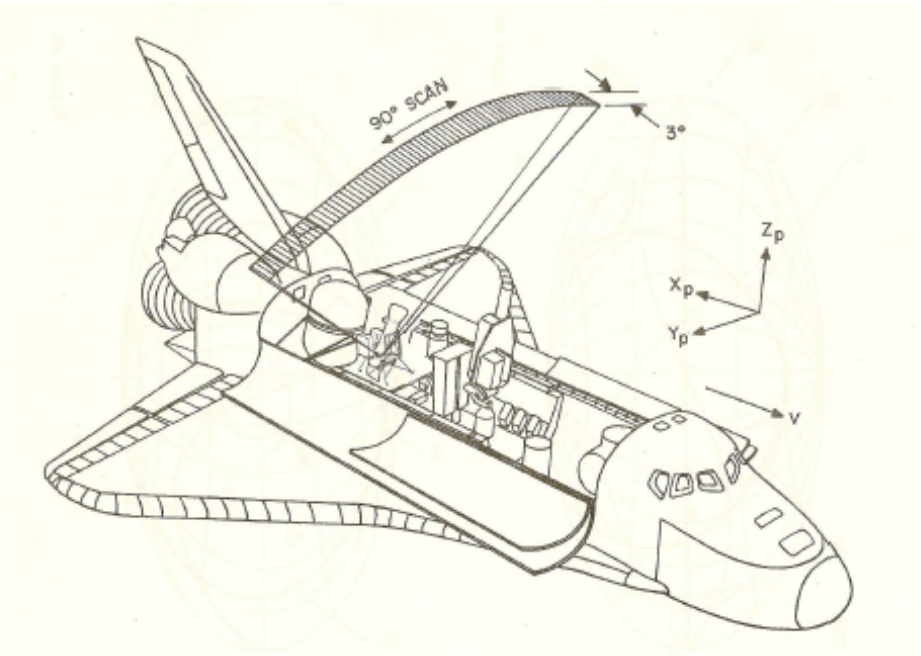
Occultation Prediction and Data Analysis

Lessons Learned

- **It’s hard to share software when most of your publications are its results.**
- **Results which are testable by observation make for reliable software.**
- **Translating software across platforms makes it more robust.**
- **Standard data formats such as FITS make data more usable across time**

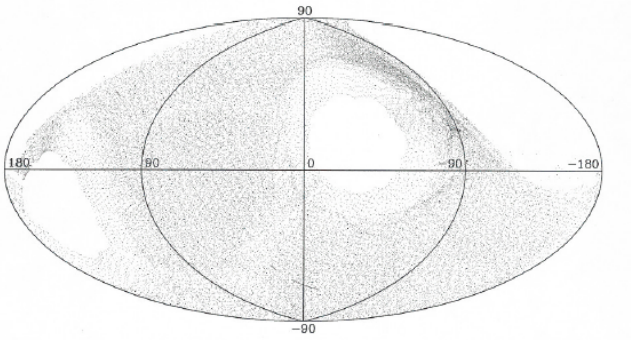
1984: Spacelab 2 IRT

IRT in Space Shuttle

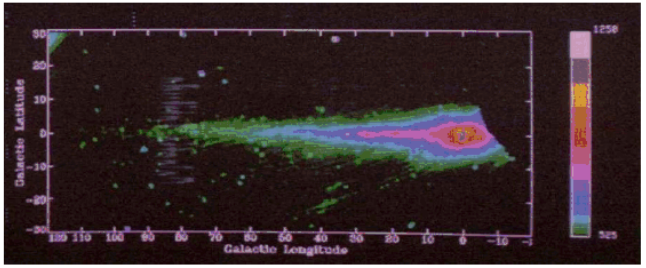


Not such a great environment for an IR Telescope

IRT Sky Coverage at 2 Microns



IRT View of Galactic Plane at 2 Microns



Add Graphics to X Window System XTERM

- **Do not add new functionality unless you know that a real application will require it.**
- **It is as important to decide what a system is not as to decide what it is.**
- **Make the system extensible so that additional needs can be met compatibly.**
- **The only thing worse than generalizing from one example is generalizing from no examples at all.**
- **If a problem is not completely understood, it is probably best to provide no solution at all.**
- **If you can get 90 percent of the desired effect for 10 percent of the work, use the simpler solution.**
- **Isolate complexity as much as possible.**
- **Provide mechanism rather than policy. Place user interface policy in the clients' hands.**

-Robert W. Scheifler and James Gettys: X Window System: Core and extension protocols: X version 11, releases 6 and 6.1, Digital Press 1996, ISBN 1-55558-148-X

X Lessons Learned

Positive:

1. Setting development goals is good.
2. If descriptive variable names and lots of comments are used, you can read and use other people's code without documentation.
3. If you need software badly enough, you can build it.
4. It helps to have external tests of your code, no matter how specialized it is.

Negative:

1. Don't buy new graphics work stations that don't have user-level graphics yet.
2. Start writing data pipelines more than six months before a mission.

Forward:

1. C is good.
2. Sharable libraries in a standard language are useful to developers

IRAF Standards and Conventions

Clearly defined and consistently applied standards and conventions are essential in reducing the "number of degrees of freedom" which a user or programmer must deal with when using a large system.
The IRAF system and applications software is being built in accord with the standards and conventions described in this document. These include system wide standards for data structures and files, standard coding practices, coding standards, and standards for documentation.
Wherever possible, the IRAF project has adopted or adapted existing standards and conventions that are in widespread use in other systems.

"IRAF Standards and Conventions", August 1983, Elwood Downey, George Jacoby, Vesa Junkkarinen, Steve Ridgway, Paul Schmidtke, Charles Slaughter, Douglas Tody, Francisco Valdes

IRAF Lessons Learned

Positive:
1. Standards are Good.
2. Inventing a new language and operating system on top of old ones works.

Negative:
1. Inventing a new language raises a huge barrier to developer participation.
2. Too many things are frozen in at the start.
3. AIPS would have worked better for us, but we weren't radio astronomers

Forward:
1. Persistent parameters are useful for users.
2. Sharable libraries in a standard language are useful to developers

WCS from IRAS Image Headers

- **Images from the Infrared Astronomy Satellite (IRAS) included code for map projections**
- **That code was added to the subroutine library I had already developed for my thesis and occultation mapping on earth and in sky plane and used to map IRT scanned data.**

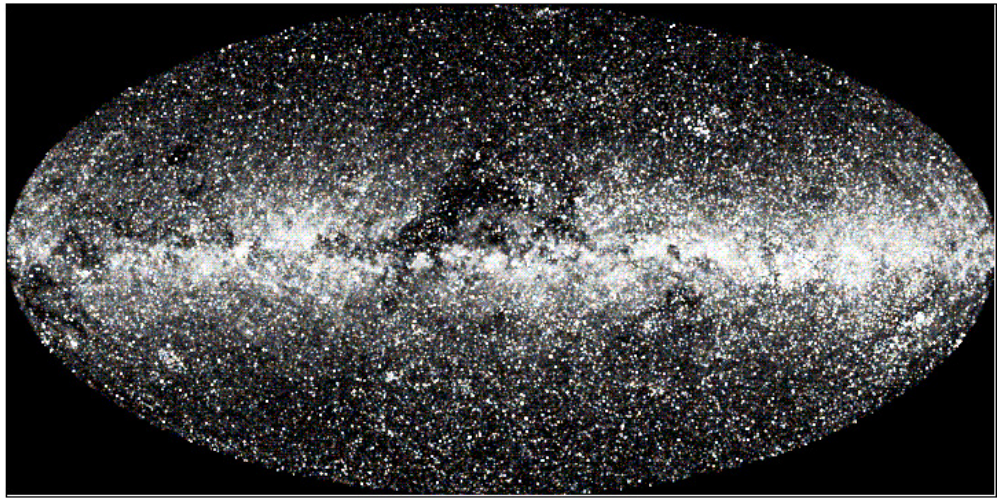
Lessons Learned about WCS

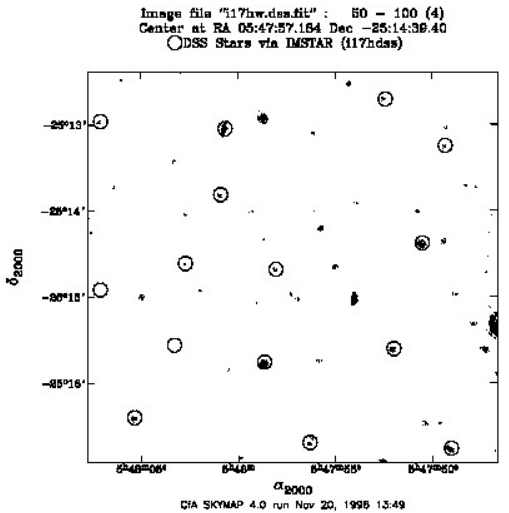
- **A standard library of subroutines with clear APIs helps a lot.**
- **Including map projection information in image headers is very useful.**

Spacelab 2 IRT Lessons Learned

- **Design pipelines while instruments are being designed, not 6 months before launch**
- **Once you write mapping software, it can be reused for totally different projects**
- **Its easier to read and write arbitrary formats of data with C than with Fortran**
- **You can mix languages if subroutine arguments are clearly documented**

SKYMAP: An unsuccessful attempt to share





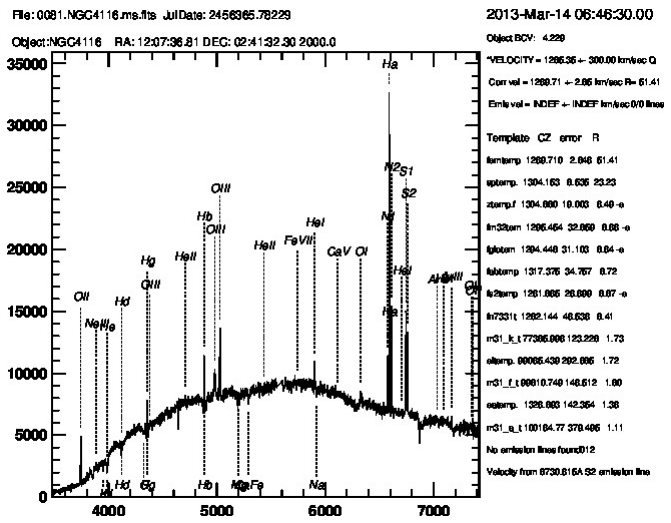
Space Telescope Guide Stars galactic coordinates

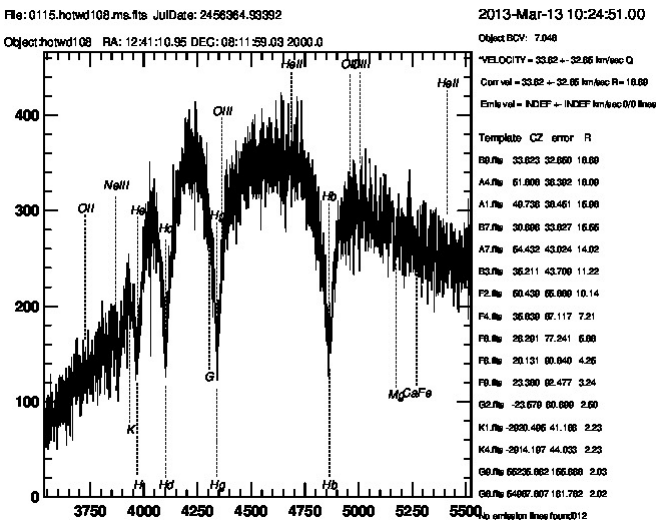
Negative Lessons

- **By 1993, ADASS seemed like a good place to show it off., but few caught on.**
- **Maybe Fortran was no longer as widely used as it was in the 1980's**
- **The combination of C and Fortran might also have turned people off.**

RVSAO: Redshifts in IRAF: 1991-present

Emission Line Galaxy





White Dwarf

- **Sometimes, IRAF is very useful**
- **A maintained software system which runs almost everywhere makes sharing easier**
- **If you use a package every day, most of the bugs get shaken out**
- **If other people use it, too, bugs get found faster**

RVSAO: Lessons Learned

Positive

- Sometimes, IRAF is very useful
- A maintained, ubiquitous software system makes sharing easier
- If you use a package every day, most of the bugs get shaken out
- If other people use it, too, bugs get found faster

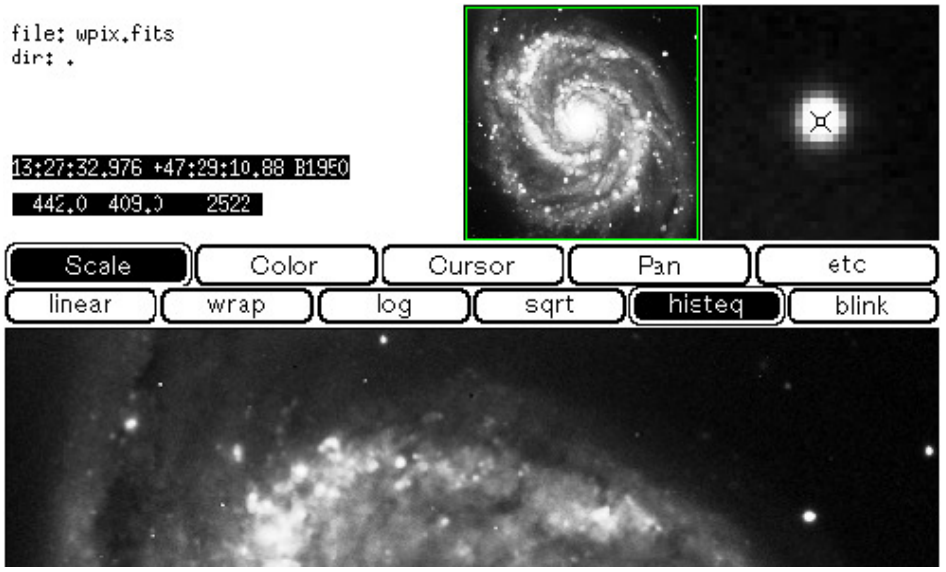
Negative

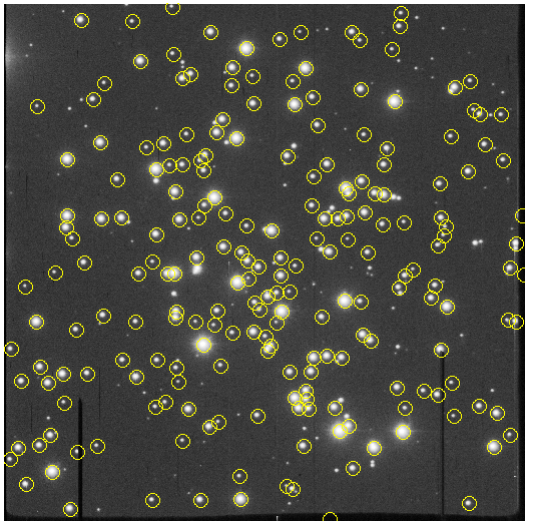
- Few others understand SPP, so the developer has to fix all of the bugs
- After 20 years the system is no longer as ubiquitous

Forward

- Write packages in a long-lived, system-independent language
- Make package as self-contained, but universal, as possible

WCSTools: World Coordinates since 1994





It all started with the NRAO release of open source map projections at ADASS in 1994
I was maintaining SAOimage, and coordinate tracking seemed like a Good Thing
Optical images didn't have good WCS mapping, but Elwood Downey wrote one
I translated my catalog and FITS handling software into C, and WCSTools was born

WCSTools: Lessons Learned

Positive

Performing services which everyone needs, but few really understand sells.
A single-language, open-source software package sells.
Good web documentation with examples means fewer requests fall on the developer.
Policy/GUI-independence helps.
Top-level subroutines with simple API's also sell.

Negative

Needs change over time.
Commonly-used processing structures raise the barrier to using external programs

Forward

Developers don't last forever, so make it easy to turn over responsibility
It helps if package developers have a long-term position and never retire.