Lessons Learned

Though it may seem minor, having an explicit software versioning scheme that is well thought-out can help to avoid a lot of confusion. For this to be an option, the third-party software must be sufficiently mature (i.e. fully developed, by avoiding software re-writes and simplifying verification and debugging.

1. Python is fine for real-time processing

Python has long been used in a "glue" role in direct broadcast software developed at SSEC. More recently, we have expanded our efforts into developing core components in Python. We are well aware of the advantages of Python (rapid software development, availability of a broad spectrum of community-supported software modules for scientific programming), but initially there were questions about the suitability of Python as a general-purpose, high-level language for software development. These concerns have been justified by testing and experience. Python has long been used in a "glue" role in direct broadcast software developed at SSEC. More recently, we have expanded our efforts into developing core components in Python. We are well aware of the advantages of Python (rapid software development, availability of a broad spectrum of community-supported software modules for scientific programming), but initially there were questions about the suitability of Python as a general-purpose, high-level language for software development. These concerns have been justified by testing and experience.

2. Re-use existing software where possible

The CSPP team has had great success in building packages encapsulating pre-existing software, often with less pain to no modifications for direct broadcast use. This strategy is particularly effective for Level 2 products, because it keeps the focus on core research while allowing the integration of existing software libraries. The strategy is to reuse existing software where possible, including libraries for reading and writing netCDF files, and libraries for computing statistics. The advantage is that the existing software is well-tested and well-documented, and can be easily updated as needed. The disadvantage is that the software may not be fully integrated with the existing software, and may need to be modified to work with the new software.

3. Put some thought into software versioning

Though it may seem minor, having an explicit software versioning scheme that is well thought-out can help to avoid a lot of confusion among team members. The versioning scheme should specify the meaning of each version number, the rules for when they are incremented, for example based on interface changes, added functionality and bug fixes. Software version numbers should appear consistently throughout software, documentation and products. We have found it to be useful to have the freedom to increment the patch number in a release, as long as the major and minor numbers are incremented in a consistent way for any given version number.

4. Use version control software, heavily

Version control systems have become a critical tool for any software development project. The provide an audit trail of what changes were made to code, when they were made and by whom. They allow recovery of old versions of code, encourage experimentation and are especially useful in collaboration development. Traditional version control systems are like any other practical tool. Primarily, the Python code base uses git, which provides a branch/merge/branch-based interface on top of git that allows branching and merging, seamless integration with commercial programs, and effective change tracking and conflict resolution. We use git also for version control, including commit history, automatic integration hooks, and wiki functionality.

5. Issue tracking + ticket reviews + code freezes

Modern issue tracking systems allow developers and support staff to track issues reported by customers, as well as bug fixing and testing tasks associated with individual software releases. Tasks can be assigned to team members, given priority and defined in terms of milestone and dependency relationships. Team members can view the status of their assigned tasks and the status of the entire project.

6. Allocate time and resources for testing and documentation

A significant amount of time and effort are needed to ensure that software is working properly before release, and to create user documentation. The testing should include all command line and configuration options, new functionality and interface changes, and automated regression testing to ensure old problems have not reappeared. Through pre-release testing often fixes bugs that are not apparent to the developer, avoiding the need for late bug fixes and release delays.

7. Make sure you own at least one machine that matches each hardware specification

A software package is tested on a number of machines before release, including a reference machine that matches the hardware specification for that software package. Testing in a reference machine greatly reduces the likelihood of bugs being introduced, as the software is tested on a system that closely matches the hardware specification.

8. Use collaboration tools

The CSPP team relies heavily on Slack, which is essentially a freeform group chat application with some enhancements tailored to software developers. Slack allows team members to define channels dedicated to specific topics, and to also send direct messages. Team members can passively monitor channels of interest, and will receive notifications when their handles or topics of interest are mentioned. Information of common interest to the team is often posted to Slack. Links are searchable, so it works as an information repository.

9. Parallel processing is required to process direct broadcast streams from geostationary satellites

The data rate from the new generation of geostationary weather satellites (Himawari-8/HG-16) is high enough that parallel processing is required to generate products in a timely manner. The software is currently at beta status. To obtain early versions of the software, contact: Graeme Martin: graemem@ssec.wisc.edu

10. Wherever possible, distribute pre-built binaries with bundled third party software

CSPP Geo software and software distribution are done binary based, with all required third-party software pre-validated and bundled. In practice we have found this method to be very effective. The advantages of this method are easy installation (tar and gunzip, and elimination of a class of user problems including build issues and software bugs due to software differences and conformance issues. The disadvantage is that the software distribution must be kept up to date with the latest changes to the software.

11. Be prepared to release rapid updates

The CSPP Geo direct broadcast streams (GRB) turned on in December 2016 and data from each instrument was added over a period of about 6 months. During this period, data was considered to be provisional quality and the priority was to get software updates to users as soon as possible to allow them to generate products. To do this, we bypassed our usual release policies in order to get new “rapid” software versions to developers as soon as possible. These software versions were developed to match specific changes to the data, and were released within a few days of the changes. This approach has been effective, and has allowed us to release new versions of the software on a regular basis.