ALMA SOFTWARE PROJECT MANAGEMENT LESSONS LEARNED

G. Raffi*, ESO, Garching, Germany
B. E. Glendenning, NRAO, Socorro, New Mexico, U.S.A.

Abstract

The Atacama Large Millimeter/Submillimeter Array (ALMA) is the largest radio telescope currently under construction by a world-wide collaboration. The first antennas (the total will be 54 12m antennas and 12 7m antennas) are being commissioned to become part of the interferometer at a high site (5000m) in Chile. The ALMA Software (~ 70% completed) is in daily use and was developed as an end-to-end system including proposal preparation, dynamic scheduling, instrument control, data archiving, automatic and manual data processing, and support for operations. The management lessons learned will be explained. Aspects described will go from requirements analysis to the use of a development framework: ALMA Common Software (ACS) in our case. The process used to provide regular releases will be outlined, including temporary cross-subsystem teams. The importance of integrated regression tests will be stressed, but also the need to validate the system with users. Among the project management tools risk analysis, earned value measures and tracking of requirements completion will be presented. Monitoring progress with reviews and the possible impact on completion dates will also be discussed.

ALMA SOFTWARE HIGHLIGHTS

Figure 1 shows the first ALMA antenna that was moved after several months of testing at 3000m to the high site (the Chajnantor plateau) at 5000m on Sep.23, 2009. The antenna is visible on its transporter, which allows not only movement to the high site but also reconfiguration of the antenna array at the high site. Antennas can move around on different pads reaching out configurations over total distances of about 40 km.

The ALMA software is an end-to-end system consisting of the following main parts:

- Proposal and observing preparation (Fig.2)
- Dynamic scheduling
- Instrument control
- Data archiving
- Automatic and manual data processing
- Support for operations

So this includes much more than the control system for the antennas and correlator, and receivers. It provides obviously the way the whole astronomy done with ALMA is organized and perceived by users.

(*) graffi@eso.org

The ALMA collaboration has three main partners in Europe (ESO), North America (NRAO) and Japan (NAOJ) with many other Institutes in other Countries associated with them. A large software group exists now also at the Observatory.

Figure 1: ALMA antenna moving up to the high plateau of Chajnantor.

The following figures characterize the software project:

- Developers are distributed over 4 continents and 15 locations. There are about 80 people concerned with software development and testing
- The present size of the ALMA software is about 2000 kLines of specially developed source code. More than 70% is complete. (Fig.3)

The ALMA software is in use for commissioning at the ALMA Observatory in Chile.

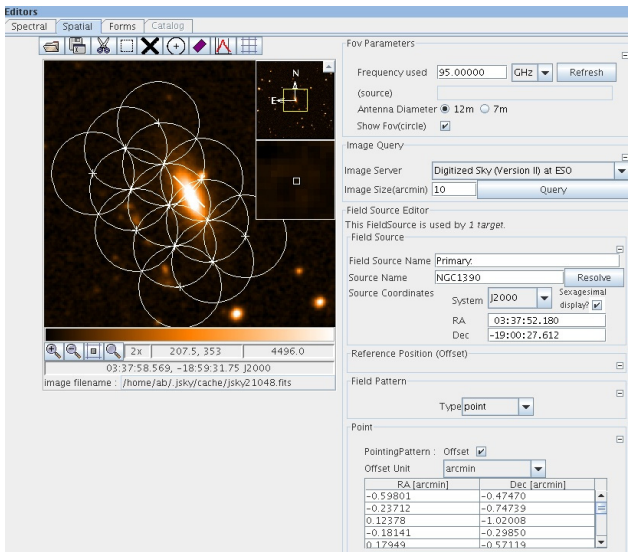

Figure 2: Observation Preparation: Spatial Editor (developed in collaboration by UK ATC and ESO).

SOFTWARE PROJECT MANAGEMENT ASPECTS

The following aspects of the ALMA software project are here considered:

- Distributed Team Management (Most important: Wiki, CVS, regular telecons, F2f meetings)
- Requirement and tracking progress
- Use of a development framework: ALMA Common Software (ACS)
  - Software written with ACS implicitly uses its architecture. This is good for distributed development to maintain consistency between different developers.
- Releases at fixed dates and Planning
- Temporary cross-subsystem teams
- Integrated regression tests by an independent team
  - And need to validate the system with users.
- Project management tools: risk analysis, earned value measures
- Reviews to monitor progress: Internal and External
- Problems reporting (JIRA system)

(*) Based on the container-component paradigm and using CORBA. The system allows the use of C++, JAVA and Python on Linux operating systems.

Paper [1] gives more details on management aspects, as seen by us one year ago.

LESSONS LEARNED

In the following section the various aspects of the ALMA software management are listed in a schematic form indicating what has been important in our experience.

1. **Requirements collection** (with Use Cases) was important. We were in the lucky situation of having a dedicated team of astronomers who took an active participation in requirements writing, including relevant Use cases. They then gave advice on how to detail requirements and in this phase the team, augmented with ALMA astronomers, performs user tests.

   Advantage:
   - There will be still missing or late requirements, but design is mostly done upfront

   Advice:
   - Requirements working group to be recommended
   - Tracking requirements completion to show progress (planned vs. actual) is relevant for future planning purposes.
2. Using a **software framework** - ALMA Common Software (**ACS)** in our case (although most of what follows would apply also to EPICS, TANGO etc) is very important.

   Advantages:
   - Allows collaborative work, homogeneous system;
   - Provides a solid debugged base of software Enforces also hardware standards and operating system versions;
   - Makes large distributed projects manageable and maintainable;

   Advice:
   - Requires team discipline and managerial support;
   - and Learning (yearly ACS courses in our case).

See also poster [2] and paper [3] at this conference.

Paper [4] gives a realistic view of the development process and effort.

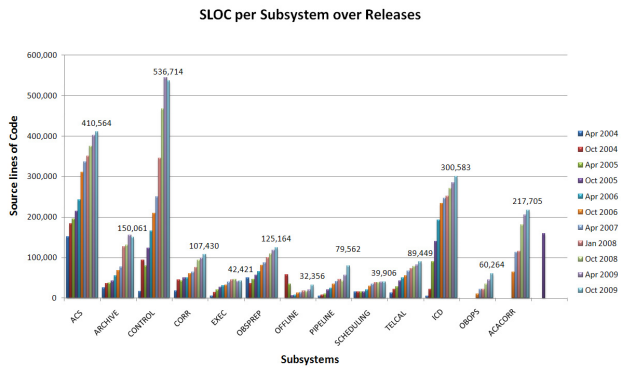

Figure 3: Source Lines of Code/Computing subsystem.

3. Incremental **Releases at fixed dates** (vs. fixed content) twice/year
   - Software is developed incrementally in 6 monthly steps (Releases).
   - Releases are an integrated e2e system
   - Patches allow to upgrade a few computing subsystems (typically 1 per Release cycle).

Advantages:
- Easier integration, predictable dates by project
- Planning work is for 6 months and can be tuned to project priorities

Advice:
- Give priority to testing and making releases over development when deadlines approach

4. Cross-subsystem **Function Based Teams** (FBTs) (~3 months)
   This is more important for a large, geographically distributed team in order to avoid an integration hell at every Release.
   Advantages:
   - Implement important functionality reducing impact of changing interfaces
   - Make integration easier, as inter-subsystem issues get sorted out continuously.
   - Integrations are more frequent, which is more important with a geographically distributed team
5. **Integration tests** (by independent team)
   Advantages:
   - In addition to subsystem tests (also by users – build-in test time up front). Avoids expensive test time at the Observatory with everybody else waiting.
   - Regression tests, eventually mostly automatic should further reduce integration tests times.

   Advice:
   - Require good test models (several computers)
   - … but cannot replace tests with real hardware
   - therefore defend towards the rest of the project the need of significant test time with the hardware and time to fix issues before software gets used
   - .. You will get anyhow all the criticism later and it will be your problem if you did not follow your procedures
6. **Problem reporting** (JIRA in our case)
   Advantage:
   - Important to track bugs/improvement request.

   Advice:
   - We are very happy with JIRA, but with any system you might have important is also the follow up.
   - We have a weekly meeting to discuss issues and flag blocking ones.
7. **Project management tools**.
   Advice:
   - **Risk analysis** helps project to assess software risks
   - **Earned Value** (apart from Requirements tracking) was difficult for us to apply in a meaningful way
8. **Reviews** (to monitor progress).
   Advice:
   - **Internal reviews** are important. We hold them yearly. Allow incremental design and also adjustment to priorities of project. Internal reviews are incremental as Releases are.
   - **External reviews** are good to prepare and comments help to see where we really are

CONCLUSION

The ALMA software is in use at the Chile Observatory since more than a year (having been previously tested on prototype antennas for more than a year) and keeps being incrementally augmented at every Release.

The procedures and organization described have allowed us to manage it satisfactorily so far. The fact of having such a geographically distributed team has undoubtedly produced some inefficiency. However this was normally counterbalanced by more rigorous discussions ahead of time every time a design choice was necessary. Frequent face to face meetings are irreplaceable to avoid later surprises.

We believe that most of what is described above can be applied to any other large project. Smaller projects also could benefit from several of the above aspects (e.g. a common framework, independent integration testing) as these help to create a more predictable product, and should help also to achieve higher quality.

ACKNOWLEDGMENTS

As the managers of the bilateral part of the ALMA Computing team we are proud to acknowledge the dedication of the Computing staff at ESO and NRAO, but also at NAOJ and all other partner Institutes around the world, who are doing a great job developing and supporting the software for ALMA.

REFERENCES

[1] B.E.Glendenning, G. Raffi "The ALMA Computing Project – Initial Commissioning" Proc. SPIE Vol. 7019-32, Astronomical Telescopes and Instrumentation, Marseille, France, June 2008.

[2] G.Chiozzi et al., "ALMA Common Software (ACS) Status and Development", ICALEPCS2009, Kobe, Japan, October 2009, these proceedings.

[3] J.Avarias H.Sommer G.Chiozzi, "Data Distribution Service as an alternative to CORBA Notify Service for the ALMA Common Software, ICALEPCS2009, Kobe, Japan, October 2009, these proceedings.

[4] J.Schwarz et al., "The ALMA Common Software — Dispatch from the trenches", Proc. SPIE Vol. 7019-32, Astronomical Telescopes and Instrumentation, Marseille, France, June 2008.