Ten things we would do differently today: Reflections on a decade of ALMA software development

Brian Glendenning[a], Erich Schmid[b*], George Kosugi[c], Jeffrey S. Kern[a], Jorge Ibsen[d], Manabu Watanabe[c], Maurizio Chavan[b], Morgan Griffith[a], Ruben Soto[d]

[a] National Radio Astronomy Observatory, Charlottesville, VA, USA
[b] European Southern Observatory, Garching bei München, Germany
[c] National Astronomical Observatory of Japan, Tokyo, Japan
[d] Joint ALMA Observatory, Santiago de Chile, Chile

ABSTRACT

The software for the Atacama Large Millimeter/submillimeter Array (ALMA) that has been developed in a collaboration of ESO, NRAO, NAOJ and the Joint ALMA Observatory for well over a decade is an integrated end-to-end software system of about six million lines of source code. As we enter the third cycle of science observations, we reflect on some of the decisions taken and call out ten topics where we could have taken a different approach at the time, or would take a different approach in today's environment. We believe that these lessons learned should be helpful as the next generation of large telescope projects move into their construction phases.

Keywords: ALMA, computing, software, development, lessons learnt

1. INTRODUCTION

Following several years of investigations, experimentation and prototyping, June 1, 2002 marked the official start of the ALMA Computing project as a collaboration between the European Southern Observatory (ESO) and the National Radio Astronomy Observatory (NRAO). In 2004 the National Astronomical Observatory of Japan (NAOJ) joined the project and finally the computing group at the Joint ALMA Observatory (JAO) was formed in Chile in 2008. A truly worldwide integrated software team resulted, with more than 12 development sites spread over four continents and involving up to a hundred staff members at its peak, on an integrated effort of about 500 FTE years. The delivered software system is an integrated end-to-end system (from proposal preparation through to distributed archiving, and automated and manual data processing) that contains approximately six million lines of source code.

Now, as we enter the third cycle of science observations with the ALMA telescope array, we take some time to reflect on well over a decade of intense software development. Rather than focusing on the many achievements that guarantee a smooth running of the observatory and stunning scientific results, we take a self-critical view on decisions that - in hindsight - could have been taken differently at the time, or would just be different in today's environment.

We are calling out ten different topics amongst the most important ones that could either have saved us time, frustrations or, of course, resources, had we taken a different approach at the time; or that would just enable us to do a much better job, should we engage in a similar or even larger project in the future. For some of the issues we have a clear alternative, for others we can only list some possible choices we could have considered. We deliberately avoid calling them the *Top Ten Topics*, because we are fully aware that such a list will never be complete, balanced and fair.

We look into all aspects we find relevant for the success of such a long-running project, not only focusing on technology choices, but also on the project management structure, requirements gathering, design and development processes, and the interactions – or lack thereof – with key stakeholders.

We believe that these lessons learned should be helpful as the next generation of large telescope projects move into their construction phases.

[*] eschmid@eso.org; phone +49 89 3200 6326; www.eso.org

2. TEN THINGS WE WOULD DO DIFFERENTLY TODAY

2.1 It's alive – keep the architecture process going

A High Level Analysis (HLA) group, consisting of senior experts with a wealth of experience not only in the design of large astronomical software projects, but also in the scientific field of radio astronomy, was formed very early in the ALMA project in the year 2000, well before the creation of the ALMA Computing subsystems. The HLA group worked very closely with the Scientific Software Requirements (SSR) group that was responsible for defining the top-level requirements for the ALMA software. The lead of the SSR team was actually also a member of the HLA group, which of course helped the collaboration. They performed an analysis and created an initial design according to the Rational Unified Process (RUP). Based on the SSR's requirements document they created use cases, sequence and class diagrams and came up with subsystem dependencies summarized in the so-called *ALMA Egg Diagram* shown in Figure 1:

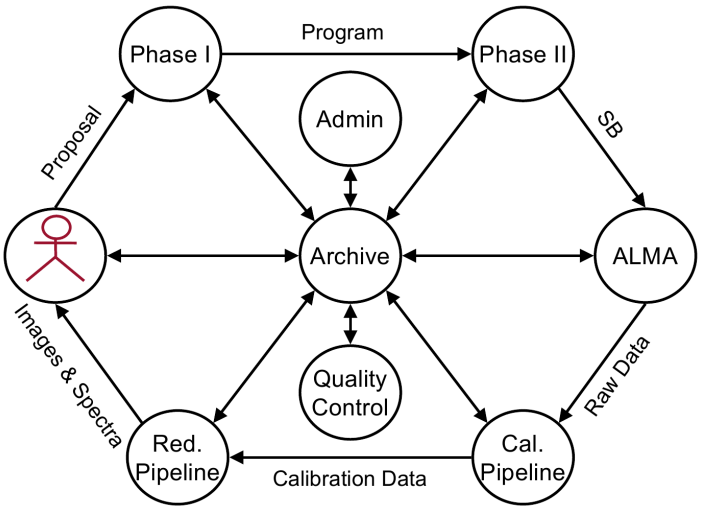

Figure 1 - The ALMA Egg Diagram

This diagram was later updated to better reflect the actual subsystem structure as shown in Figure 2, but the overall architecture is still valid today (see also [1]).

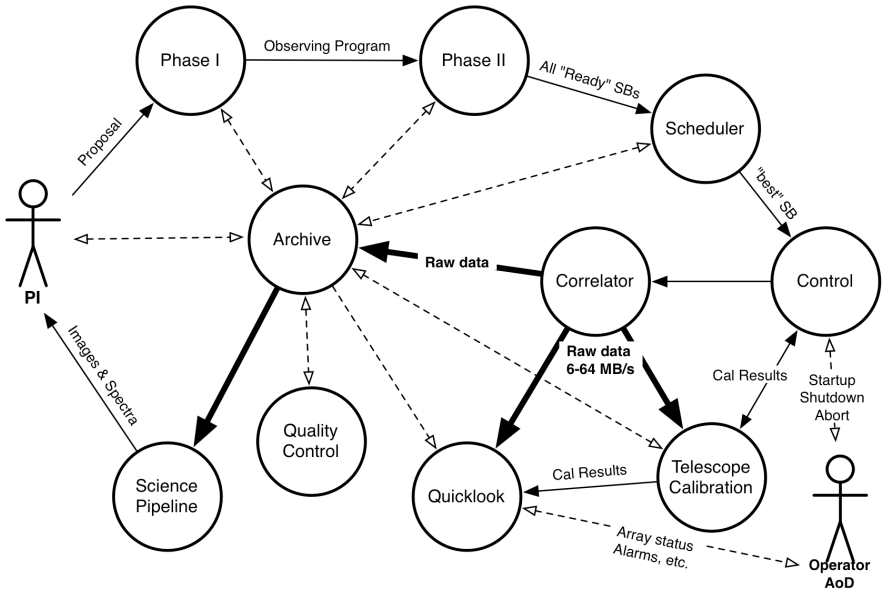

Figure 2 - The final ALMA Architecture

When the subsystem development groups became reality and started their work, the HLA activities were ramped down and most of the original members left the HLA group. But it was evident at that time that a proper connection between HLA and the subsystem groups was still missing. The subsystem groups began designing their internal structure, but the over-arching ALMA system structure was lagging behind. To some extent this was due to already having made design and implementation decisions with the development of the ALMA Common Software (ACS) parallel to the HLA work. It was clear that CORBA containers and components should be used and how communication would have to be set up. These elements were thus not discussed further in the HLA, and maybe this is why some analysis from the overall level into the subsystems was not performed. It might have led to different technology choices considering some details such as data rates and sender / receiver structure as it is now.

Starting in late 2005, the HLA team also took care of overseeing the work of the many Functional Based Teams (FBT) that were assembled to implement specific functionality requiring strong coordination between many subsystems. The last contribution from HLA to the project were the Human-Computer Interaction (HCI) improvements initiated in response to a recommendation coming from an ALMA Computing internal review held in November 2008.

The HLA team was very active until 2004, the time at which most of the ACS functionality had been developed. Subsequently the effort was gradually reduced from the original maximum of 6 people to less than 0.5 Full Time Equivalent (FTE) in 2008. From 2011 onwards the group was discontinued altogether.

While the HLA effort was being reduced, first system integration efforts, where hardware and software began to be used together as a system, started in 2007 at the ALMA Test Facility (ATF), followed by activities at the Operations Site Facility (OSF) in Chile from 2008 onwards. Since then the ALMA software has been used routinely to reach different project milestones and the number of users, and also different types of users, grew steadily from year to year.

This phase introduced a growing set of (additional) requirements coming from integration, commissioning and operations activities that were not entirely captured by the SSR group in the previous years. Each set of requirements became at some point in time the dominant one, revealing that some design aspects still needed additional development. Furthermore, these new requirements were often targeted at one particular sub-system or tool, though they were in reality system requirements. The lack of the HLA team meant that separate sub-system teams had to make this realization themselves and initiate the system analysis across the teams.

The presence of a Chief Architect or HLA group as a whole would have made it easier to continue the decomposition of new or modified system features or to organize a coherent involvement of key developers from different computing subsystem groups contributing to it.

A couple of examples where continuing the role of an architect would have been useful are:

> a) the development of the archive functionality, which should have been subdivided further during the initial testing period at the OSF rather than preserving a unique central archive system that needed to cope with very different, and sometimes orthogonal, needs; and
>
> b) the logging, monitoring and alarm systems that needed to be adjusted to make them better suited for operations as the number of array elements started to grow and experience with the production hardware was gained.

An example of the need of close interaction between software architects and developers was the contentious relationship between the ACS and Control subsystems. The ACS subsystem group was responsible for providing a common software infrastructure, largely based on a container/component model, and a number of pre-coded design patterns and libraries to be used by the application developers of each subsystem. The Control subsystem group, on the other hand, was a key client of ACS, using a significant fraction of the facilities provided by the common software infrastructure. As in any producer/consumer relationship, tensions emerged between the groups from the very beginning, aggravated by the fact that both teams were largely located on different continents, eight time zones apart (EU and NA, respectively).

Senior management promoted duty trips back and forth between both sides of the ocean, but although in some cases there was significant progress, like for instance the development of the threads library, this strategy was not as successful as expected. A better approach would have been to assign more often staff to work on the other side of the fence for extended periods of time (for instance, six months to a year). This could have helped not only to allow the end client to

contribute to the design of solutions, but also to allow software infrastructure providers to experience the client's perspective. In fact, this strategy was used successfully after the early recruitment of operations staff in Chile, where most of the new hires spent about a year working at the ATF to build up experience.

In summary keeping a Chief Architect and a dedicated group such as the HLA group until the end of the construction project, and beyond, would have helped to continue a uniform development of the design once operations requirements started to appear. Having staff going back and forth from each side of the fence would have definitely helped to facilitate the design development and mature solutions implementation.

2.2 There's only one ALMA array – wishing for improved simulation capabilities

Simulation capabilities are key for software development, integration and verification activities, not only at the time when the operational hardware is not yet available, but also during routine operations, when access to the operational hardware for testing purposes is scarce. In ALMA, simulation capabilities were initially developed to satisfy Control and Correlator subsystem needs, supplying them with virtual hardware devices to interface with the software components being developed. Additional simulation layers and capabilities were added during the years by different teams to include simulation of the single field interferometry use case, higher number of array elements, enhanced correlator simulation capabilities, generic ACS components simulation capabilities, code generation of simulated control subsystem components [2], etc.

However, the incentives for developers to make intense use of simulation capabilities were low in the early phases of the project, as the first simulators were incomplete and required specialized real-time machines to execute (see 2.8.). Meanwhile, integration with the early hardware took priority and pushed dedication to improving simulation capabilities further ahead. For example, the Control group was busy with the prototype ATF antennas until 2008, while the Correlator group had at least one actual correlator quadrant at the development site until 2012. Only when the test hardware disappeared simulation capabilities gained significant attention at these sites, leading to rapid progress towards a high fidelity hardware simulation, flexible and lightweight enough to effectively support development and testing. Overall, simulation was not seen as a core computing deliverable, but mostly as an intermediate tool to make progress in the initial development phases. No computing subsystem group was in charge of the overall simulation architecture to ensure that capabilities were added consistently and in anticipation of future development needs. For instance, a (very limited) simulation of the full 66 array elements was only available in 2011. At the same time software scalability issues were being investigated that had already shown up during commissioning activities [3]. But access to the real antennas, in particular for lengthy performance and stability tests, became a lot more difficult due to conflicting needs of the science and engineering groups.

In retrospective, a better approach would have been to add simulation capabilities to the initial work breakdown structure, considering it part of the overall architecture and making it an official deliverable of computing. Resources from different subsystem groups should have ensured that simulation capabilities were available well in advance before they were actually needed. This would then have enabled us to perform software verification activities largely in simulation, which in turn would have minimized the need for frequent access to the production environment during the integration and commissioning phase, substantially decoupling software development and testing from the rest of the activities at the ALMA site during this period.

This is of course easier said than done, as the main driver in the last years of the construction project was the addition of new software features needed by the commissioning team to demonstrate array capabilities. Proceeding in this direction would have also required clear support from ALMA senior management, which would have needed to continuously monitor the right balance between these two aspects, to ensure that both short term (new capabilities) and long term (minimal use of the array for software verification) goals were equally fulfilled. There is of course also a cost factor to be considered. Developing very good simulation capabilities takes time and resources, and additional hardware is required as well. But overall we believe that the additional investment would have paid off in the long term, by spending less time in analyzing and addressing issues on the operational system, making it cost-neutral, if not an actual saving.

This lesson has been incorporated, for instance, within the ALMA Phasing Project [4], which is one of the first ALMA development projects, whose main goal is adding Very Long Baseline Interferometry (VLBI) observing capabilities to ALMA. In this case, the design has included right from the start an important focus on simulation capabilities to ensure that as much of the functionality as possible could be verified in advance.

2.3 Keep it simple – things become more complicated anyway, but never simpler

ALMA is a complicated telescope in all senses: long-baseline interferometry, compact array interferometry, single dish observing, more than 10^5 monitor points, integrated southern/northern hemisphere operations, automated scheduling block based operations, …. While this fundamental complexity in the observatory would necessarily always result in a complex software system in terms of architecture, the mindset that resulted was applied at lower levels as well. This tendency was aided and abetted by a pervasive tendency in ALMA stakeholders (operations, engineering) to always insist that complicated use cases were essential, and were needed to be delivered early (see 2.9).

For example, the ALMA operational alarm system (for alerts during observing) has many in-principle useful features:

1. Publish/subscribe mechanism so that alarm creators and consumers can be decoupled, e.g. allowing new devices or display panels to be introduced as *only* a configure-time update
2. Several different severity levels
3. An *audience* identification (software developer vs. operator vs. astronomer on duty …) of log messages
4. Options for storage of log messages in different locations (operational archive, disk files, memory only)
5. Facilities (*reduction rules*) for automatically throwing away or hierarchically grouping messages, the idea being to suppress (but optionally having available) the cascade of error messages that result from an anomalous situation (out-of range power supply invalidates several devices which invalidates a data stream which …)

While in isolation all of these features sound attractive and sensible, the end result was that far too much effort was spent on tuning and configuring the alarm system itself rather than tracking down issues with the underlying emitted alarms, resulting in far too many alarms being displayed (even with low-priority messages filtered) on the operator panels, which overwhelmed the operators who came to ignore the alarm panel, which in turn caused real problems to be overlooked, which in turn led to episodes of lost data or observing time.

While all of these features were in response to user requests, in practice ALMA would have been better served by starting with a much simpler system, concentrating on the system engineering to understand and alleviate the underlying hardware alarms, and then building up features in the Alarm system based on prioritized requests arising from real-world use, not theoretical use cases nearly 10 years old.

We note that the Alarm system was largely reused from another flagship scientific enterprise, who subsequently abandoned it.

Another example is the ALMA *Science Data Model* (SDM, also shared with the VLA). This is the ancillary and descriptive data which goes along with the bulk data that allows the telescope data to be processed (calibration, flagging) and imaged after the data has been observed. This is the fundamental data interface between telescope observing and data processing (pipelined or manual). In terms of dataflow, the SDM is created by the ALMA online system, stored in the Archive, and then is retrieved and converted to the format of the post-processing system (normally a CASA *MeasurementSet*).

The ALMA SDM consists of about 60 table types (with many internal linkages), the CASA *MeasurementSet* consists of 18, and the FITS IDI software that is used to record radio interferometry (typically VLBI) FITS data has 16. It is very true that the SDM is a better model of a radio interferometer than those previous efforts. And it is also true that some of the tables are used for recording online telescope calibrations which are not needed in post-processing. But nevertheless the ALMA SDM is considerably richer than it needs to be to fulfill its fundamental mandate: providing the data needed for post-processing. This resulted in a considerable development effort in the online (control) software's *data capture* component to assemble this data, and within the CASA format converter (*filler*) to essentially undo that complexity and put it into CASA. The combined effort was more than 10 FTE-years. Had there been less of an *impedance mismatch* much of this effort could have been saved, and the commissioning/debugging could have been much easier for the overall project.

The general lesson learned was that you can always add complexity later when warranted, but it is very hard to subtract it.

2.4 Take all factors into account – why XML databases failed on us

Oracle XML DB was introduced in 2001 in Oracle 9i and the features have gradually been extended. It gave an Oracle Database the ability to store XML documents as special XML types or *shredded* relational tables, for querying using XPath and XQuery, for indexing the documents, and for manipulating the documents in the database.

The decision to use XML databases was a sensible and logical conclusion, given the prevailing technical fashion of the time and the fact that our principal data interchange format is in XML.

Some of the facilities offered by Oracle seemed rather exciting, like the option of storing of XML in dedicated columns with the underlying storage being either plain text or an optimized prepared binary format. In addition to that the possibility existed of telling Oracle to automatically *shred* the documents and store them in traditional relational tables.

The storage of XML documents allowed the Archive to be easily built up while the data schemas were still in a process of rapid flux without fixing a schema.

Our first hurdle was with the XML storage mechanism. We operate in an environment where Oracle Streams replication placed additional constraints on the data types we could use. We could only store XML as text. As XML can be verbose, this resulted in additional IO and storage space overheads. For the majority of our documents, where content far outweighed markup, we had an XML markup overhead of around 10% of the total data size. For the worst cases we had documents where the XML markup added around 400% to the data content. Wasted space costs money in terms of infrastructure and eventually operational problems such as backups.

The approach was manageable at first.

The documents were stored without schema validation. To the more visionary amongst us this was a huge bonus. The Archive could and should store all documents, regardless of content, provided they are well-formed. Validation should be performed by the software, allowing more complex rules to be coded. For the more conservative in the team it was a disaster waiting to happen. Database-level integrity checking provides a safety net to ensure that your data is consistent - the fundament of any robust system. Although we could have enabled schema validation, we could not check things a traditional relational database offers, such as referential integrity between documents. Indeed we have been bitten by software errors allowing erroneous documents to be stored which would have been caught in a relational database.

Schema updates were a large operational problem, although partially self-inflicted. Because we *can* store any XML documents the temptation has always been to allow storage of multiple versions in the same archive as the schemas evolve. To prevent all the software having to cope with all versions we instead decided to keep a single version and migrate all XML documents to the latest version using XSL transforms in the database. Schema versioning is also non-trivial in a relational database; however, with XSL transforms taking place in the database every single XML document had to be read, parsed, transformed, and re-written, hugely increasing the duration of the update. The problem has been made worse by Oracle licensing leading to processors being removed from the servers to keep costs down. The risk of a bad transform was terrifying, and the overhead of testing was large.

The real issues surface when we want to perform cross-document operations, such as searching or exploring the database content. Considering the work they are doing, Oracle's searching facilities are magnificent, but still far too slow for our needs. First we had the hurdle of having to learn XQuery. While most of our developers are proficient with basic SQL, this was a big drain on effort with many gotchas to master. The ability to index XML (which essentially creates a relational table holding the data of interest as an index) proved to be problematic, too. Eventually developers worked around the limitations by creating their own relational copies of searchable data using triggers or external harvester code, often creating multiple copies of the same data for different purposes.

Our schema designs have long since stabilized, and it turns out they are mainly structured, eliminating the driving argument for flexible XML storage; we are effectively storing XML in the database because it is convenient. We are also upgrading to Oracle 11g r2 and looking at replacing Streams with Golden Gate. This would open up new possibilities to us if we insist on keeping XML stored in the database. However, we have now reached the point where we are looking at switching to relational storage instead. Given the costs of Oracle licenses we are not considering using Oracle's XML shredding, which would limit our future possibilities.

One can of course not simply say that all would have been good if we had chosen relational storage instead right from the beginning. Surely there would have been other issues. But we would be very careful to fully evaluate all aspects of a very promising looking new technology, before applying it, especially in such a core area.

2.5 Go mainstream – avoid niche solutions

The computing portion of the ALMA construction project lasted more than 10 years. Inevitably over such a long timespan some adopted technologies did become less popular and moved into niche status. But nevertheless the original choices were reasonable given their prospects at the time the decision was made. Examples of this in the ALMA software stack include the use of CORBA middleware and Java for desktop applications.

However, there were also examples of technologies we adopted that were at the time obviously niche solutions. While we could (and did) hope for increased adoption in the open source community or for commercial support, we should have known that we were taking a gamble. And, while it is true that an open source license means that you can always support it yourself, taking over someone else's large body of code, which only has a small user base, means that you are also taking over a large corpus of bugs compared to a narrowly tuned code supporting exactly your requirements (hopefully the minimum needed, as described in section 2.3 above). We include in the set of technologies we should have avoided the following:

1. We used an open source implementation of the CORBA Audio/Visual streaming service to transfer bulk data from the correlators to their clients: the online calibration software, quick look display software, and the archive. Developed as part of a university research project, we believe that shortly after it was adopted, we were its only notable user, and moreover we were using it outside of its primary domain (audio/visual media transmission with QOS). In test environments it worked well enough, but in the field it was prone to freezes and other erroneous behavior in the face of occasional, hard to reproduce, system and network glitches. After considerable debugging effort we ended up replacing it with a commercially implemented and supported Data Distribution Service (DDS). This was money well spent.
2. In the early days of ALMA we used a commercial real-time operating system (VxWorks) to support initial test setups. We then decided to switch to a Linux based solution, which was in principle a good choice. However, we chose an academically supported real-time Linux kernel with a commercial version of Linux (Red Hat) layered on top. Although the community for this kernel was enthusiastic, it was small and not set up for support. We had a regular set of problems related to third-party drivers for uncommon hardware boards (we typically had to re-engineer them), and more importantly operating system upgrades (i.e., Red Hat version updates) often encountered problems that took a long time to resolve. In retrospect we would have been better off not using any version of real-time Linux as most of the advantages have faded away or are marginal, but rather structuring our control system as normal Linux applications and normal device drivers, an approach which has been used successfully at other observatories.
3. The AIPS++ package (predecessor to CASA, ALMA's data reduction package) adopted the Glish scripting language as its internal scripting language. Glish was developed as part of the Superconducting Supercollider (SSC) project. Unfortunately by the time it was adopted by AIPS++ the SSC had already been canceled and it was apparent that the AIPS++ team would have to maintain Glish. This maintenance effort alone consumed a full FTE. While Glish was elegant and perhaps unmatched in some respects for a scripting language (e.g., asynchronous event programming), its lack of a vibrant development community meant that it would always be running a distant second in terms of platform support, bound graphics and numeric libraries, and user documentation. Glish was replaced with Python as part of the transition to CASA about 10 years ago, losing more than 100,000 lines of Glish application code that had been developed. It is however worth noting that when Glish was selected, Python itself was not well developed, and e.g. had no numeric library. It is therefore quite possible that Perl or Tcl/Tk might have been chosen instead, which probably would have come with their own problems.

2.6 Many small pains are easier to deal with than few big ones – deliver software more often

ALMA software releases are a compound of several functional subsystems (e.g. Control, Correlator, Archive, Scheduling, Telescope Calibration), each of which contains several software modules providing both, the subsystem-internal and cross-subsystem functionalities.

At the beginning of the project, the software releases were delivered every six months, including a lot of new functionality for each subsystem. After integrating the deliveries from each subsystem group, these releases were tested by using simulation capabilities and later with prototype hardware at the ALMA Test Facility [5]. The process of stabilizing the releases took a long time due to the large amount of changes introduced simultaneously in all the subsystems. This also generated many sources of problems. Several iterations of a software release, which included patches for fixing problems detected during the testing, were necessary to reach an acceptable mature software level. Thus, the integration and testing time spent only for reaching the same functional level as the previous release was about two months on average, without even considering the testing of the new functionality.

The construction of the observatory in Chile and the decommissioning of the ALMA Test Facility in the US introduced new challenges for the software delivery process. The access to the operational hardware in Chile was very limited, because it had to be available most of the time for science commissioning activities rather than software testing. On the other hand, software simulation was not mature enough for full verification of the releases, so the testing team strongly depended on the availability of the operational hardware for completing the testing of an ALMA software release. Two months dedicated to integration purposes before delivering new capabilities was definitively not acceptable for the observatory.

In order to revert the situation, in 2011 ALMA Computing management took the decision of introducing some modifications to the software delivery process. At this point, the model used by ALMA was strongly influenced by the traditional *waterfall* model, where features were planned according to the requirements received from stakeholders (engineering and science groups) and implemented in a development branch according to the plans of every subsystem group. After all subsystem groups completed implementation, the integration and testing team created a new integration branch (derived from the development branch) and proceeded with the verification process. Once verification results were acceptable for putting software into production, it was delivered to the stakeholders for the final validation. As we mentioned before, this process took about six months of implementation plus two months of integration and testing before delivery to stakeholders.

The main change introduced in the process was the implementation of *incremental releases*. This follows basically the same model as before, but adding some agile elements. Software releases were restructured into smaller cycles, which included software implementation, verification and integration. Thus, three phases were identified and defined as part of the process: Phase A: Implementation and Developer Testing, Phase B: Verification and Integration Testing, Phase C: Software Validation. The first two phases are under the responsibility of the computing group and the last one is the responsibility of stakeholders as the receivers of the software releases. The whole cycle for the first two phases was estimated to be about two months, including the testing of new functionality. The scope of each software release was also reduced and ordered. The idea was to provide the same functionality over the same period of time, but deliver it across multiple incremental releases. This approach also optimized the use of computing resources in terms of development and testing, since the schedule for the incremental releases allowed a better organization of the two phases managed by the computing group (as shown in Figure 3).

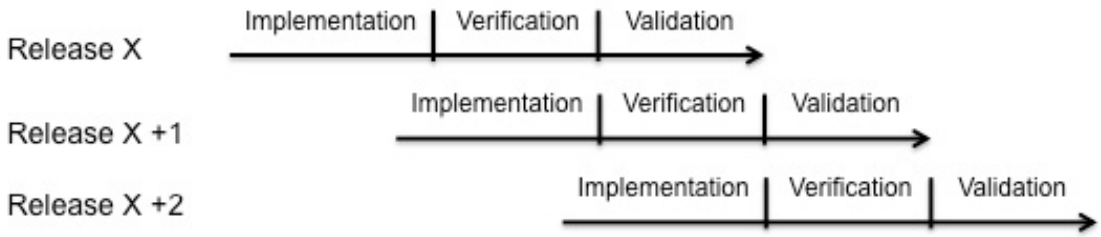

Figure 3 - Release cycle process

One of the main problems that this approach was faced with was the resistance of stakeholders of having more releases to be validated, since they had the perception that the workload on their side was increased. However, this perception disappeared after the delivery of some incremental releases, when it became apparent to them that the same functionality

was being delivered, but in a more organized and structured way. Even more, the advantage of having a concrete and reliable release schedule was also useful for them in order to distribute their resources for the validation phase.

Probably the incremental model would not have been very useful in the beginning of the project when the software development had just started and there was no observatory to receive them. At that point, there were no strong requirements for having functionality delivered quickly into production and access to the prototype hardware was readily granted for testing purposes. However, the construction of the observatory triggered new requirements and challenges as explained above, and the software delivery process had to be modified to fulfill those requirements. In that context, the idea of delivering small incremental releases seems to be more adequate and efficient than big deliveries. With hindsight, the switch to more frequent incremental releases should have been introduced earlier, coinciding with the start of science commissioning activities at the observatory. It could have helped avoiding a phase when the ALMA software was seen as a main obstacle in getting ready for the start of Early Science observations.

Of course, the current process must be continuously improved as the operational processes at the observatory mature, but this was, for sure, an improvement from the computing point of view.

2.7 Testing ain't easy – don't believe that one small team can do it all

One of the most challenging aspects of developing software for a giant and complex instrument, such as the ALMA observatory, is that most of the software needs to be written at a time when the platforms it will eventually run on merely exist on paper. While careful planning and design allowed us to successfully develop much of the code and test it in isolation, e.g. through unit testing, the integration of various subsystems and integrated testing proved to be much more of a challenge.

A relatively small group dedicated to Integration, Testing and Support (ITS) was created early on in the project. The team has always been fairly evenly distributed around the ALMA partners with a proportion of developers versus testers of approximately ten to one. Their initial focus was on establishing a Standard Test Environment (STE) mimicking the eventual operational environment. Instances of the STE have been deployed at all major development sites and eventually also at the observatory in Chile. The STEs have been used by developers and the test group itself ever since for initial verification of newly developed features, as well as by the testing team for integration and regression testing.

Initially, when the targets for deployment were the ALMA Test Facility (ATF) and indeed the STEs themselves, things were going reasonably well. This was helped by the fact that developers and scientists were keen on trying out the integrated software for the first time and did not mind experiencing issues along the way, which essentially meant testing the software, of course. There was also a strong focus on the so-called online software, i.e. the software used during live observations, at the time, while much of the offline software, i.e. the software used to prepare and submit proposals, tools for proposal review and observing project generation, numerous operational tools, as well as the data processing software was still in earlier stages of development.

As the software matured and increased in scope, things became more difficult. Also, with the observatory in Chile becoming a reality and more and more antennas being added to the array, the focus shifted to Commissioning and Science Verification (CSV). This meant that a dedicated team of commissioning scientists expected a stable version of the software to focus on their scientific tasks at hand. Testing of software was seen as a distraction from the commissioning activities. At the same time much of the offline software was getting delivered and needed to be integrated and tested as well. The size of the ITS team, however, did not increase. Nor was the required skill set (e.g. to include more scientific expertise) of the testing team reviewed at the time, and, as mentioned in the previous section, the goal remained to deliver all software subsystems in one big version. All of this led to a situation where the ITS test team worked very hard to integrate and test all new features – and there were a lot – in a short period of time, run the required regression tests and develop new system level tests at the same time. This mission was impossible to achieve and hence frustrations started to show up everywhere. Finally with the start of Early Science observations being imminent in late 2011 and with it the need for stable software further increasing, action needed to be taken.

ALMA Computing got together with representatives of all key stakeholders, such as Science Operations, CSV, Engineering and the ALMA Directors to define an all-encompassing and mutually agreed delivery plan for the ALMA software, that clearly defined five discreet testing phases: developer testing, ITS integration and testing, software verification at the observatory using as much as possible the operational hardware, commissioning and scientific validation and finally acceptance testing for science operations. In addition the roles of a software Release Manager, who leads the delivery planning on the computing side and an Acceptance Manager, who plans the software acceptances on

the observatory side were created, and *subsystem scientists*, i.e. scientific experts assigned to oversee each computing subsystem, play an important and active role in the validation of software features. All phases of testing are since then fully tracked in the JIRA issue tracking system, which allows all involved parties to plan their activities reliably.

Overall the process is working fairly smoothly now and, very importantly, all the processes are being reviewed and fine-tuned on a regular basis to not only correct small inefficiencies, but more importantly to keep track with the evolving conditions and requirements of the observatory as we move into regular operations. With hindsight, we would of course have taken the necessary steps to ensure a smooth software delivery process at an earlier stage.

2.8 One for all doesn't always work – hardware access must be easy

Access to hardware is a very important aspect of any kind of control system and especially if real time behavior is part of the requirement. How to balance the usability, security and performance is fundamental to success in the commissioning and operations phase of a project.

The main hardware access protocol in ALMA is based on the Control Area Network (CAN), a well-proven standard that is extensively used by industry, especially in the automobile manufacturing area. ALMA software has implemented additional features in this bus: a timing event and a reset signal. On the driver side, proprietary kernel modules based on RTAI were developed to read and write CAN messages in the bus. Messages specific to each piece of ALMA hardware are clearly defined in its corresponding Interface Control Document (ICD).

The minimum setup required to communicate with hardware located in each antenna is to boot a diskless real-time computer, and load the image with the real-time kernel and preconfigured *ramdisk* with the operating system. After that, the kernel modules should be loaded and then applications in user space can communicate with the kernel modules through dedicated FIFO named pipes. A C/C++ class named *AmbInterface* was developed for this purpose. Other classes with additional features use composition to add more functionality to the one defined in the *AmbInterface*.

At this level, in order to communicate with the hardware, a clear understanding of the specific ICD is required and bitwise operations are fundamental to compose a CAN message and parse the returned data. This is the lowest level access point to the Monitoring and Control (M&C) channels, in which a Python based component named *AmbManager* is created to allow the simplest, and self-contained method of access to the hardware. On top of this minimum setup, high-level classes, which hide the bitwise complexities, were created for each hardware device as part of the Control subsystem. This subsystem needs a number of other services in order to allow centralized access to processes in a distributed environment (services such as naming services, logging, automatic monitoring, data retrieval and archival, alarms, all mostly coming from the ACS subsystem). The possibility to abstract from the distributed nature and accessing the hardware in a centralized way notably simplifies the logic required to develop radio astronomy observing software. Multiple access points allow commissioning activities and hardware maintenance activities to be executed efficiently and in parallel during the construction and commissioning phase of the observatory.

After operation started in the observatory, soon, a missing use case was detected: how to access to the M&C channels when the whole software infrastructure is not up and running, i.e: when a newer version of software has to be deployed for testing purpose. This situation makes hardware maintenance activities difficult to schedule in parallel with the software maintenance activities, reducing the overall efficiency of the observatory in operation. Although, this use case was never specified as part of the requirements, in retrospective, a way to separate hardware access from other centralized software services could have been implemented.

After operations started at the observatory, soon a missing use case was detected: how to access the monitoring and control channels when the whole software infrastructure is not up and running, e.g. when a newer version of software has to be deployed for testing purposes. This situation makes hardware maintenance activities difficult to schedule in parallel with software maintenance activities, reducing the overall efficiency of observatory operations. Although this use case was never specified as part of the requirements, in retrospective, a way to separate hardware access from other centralized software services should have been implemented.

2.9 Commissioning comes before operations – transient and steady state are very different!

Although this statement seems self evident, the implications for software development may not be. In the case of the ALMA Computing system development our failure to recognize and plan for this led directly to issues at several epochs in the construction project.

In the ALMA case we had a distributed team working on all aspects of the telescope simultaneously, focused on the final operational model: scheduling blocks, created by the observing tool, selected and queued by the scheduling subsystem, executed by the monitor and control system, reduced by telescope calibration and stored in the archive for later analysis.

At the same time, early commissioning work was focused on evaluating the fundamentals of telescope motion, ensuring that the telescope could meet stringent requirements for pointing accuracy and settling times. Not having planned for these two objectives to be pursed simultaneously led to several sub-optimal results. Both the users (the very first clients of our new software system) and the developers became frustrated by a software system that attempted to satisfy both needs but did neither well. One result was a forked version of the observatory software known as TICS (see [6]) that was developed for the use of the early commissioning team, but was an evolutionary dead end in terms of the long-term development of the ALMA system.

The same issue occurred later in the project as the prototype back end hardware was being commissioned, the monitor and control system designed for steady state operations proved too inflexible for the very dynamic use cases of early integration. Variations for the vagaries of prototype hardware (often multiple hardware revisions co-existing), changes in monitoring requirements for different commissioning activities, and flexibility for the needs of engineering process development were beyond the abilities of our fledgling software. The result this time was another dead-end spur of software development, but this time completely outside the control of the ALMA Computing team. Details of hardware algorithms and work around solutions for known problems were developed in an engineering interface completely opaque to the computing team. Rediscovering this knowledge cost the project time and effort, and removing this parallel system from operations once established was a difficult and lengthy task.

The usual software development approach is to do the easiest and most common things first and then come back to address edge cases and more difficult use cases. This places the computing team priorities in direct conflict with the needs of the commissioning team. As a concrete example it is easiest for the computing system to first perform all system configurations serially, and then return to parallelize operations to increase efficiency. However for the ALMA project, many of the key requirements revolve around fast time scales, or require the highest data rates to be effectively tested. This mismatch between the natural computing priorities and those of the commissioning team has been an issue throughout the construction project. The ALMA project management identified this manifestation of the problem and close coordination (weekly meetings for the last 7 years) between the two teams has mitigated this issue considerably.

The final tension between the needs of the software development team and the needs of the commissioning team is stability. This issue arose both in the early era of ALMA when TICS was created and later during the integration of the array. The fundamental conflict is between the demand for additional capabilities and progress toward the eventual operational module, and the desire to have a stable software platform so tests can be repeated throughout the acceptance process. In ALMA the relatively long time scale of the acceptance process exacerbated this issue. In the beginning of the project this was addressed by a separate fork of the software (TICS) during the later phase of the project this was eventually resolved by the acceptance team using a frozen version of the software, with the associated maintenance costs.

It is not clear that these conflicts can be entirely avoided; however recognizing that these issues will occur and planning in advance can certainly decrease their effects throughout the project.

2.10 Look who's looking – GUIs are for the users

In late 2008, a review panel in charge to "… *carry out a comprehensive review of ALMA Computing and its schedule for completing all major milestones*" noted in their report "*the non-uniform implementation of Graphical User Interfaces (GUI), even within the same sub-system.*" The review panel expressed that "*especially for Operations the layout and intuitive nature of GUIs are important*" As a result, ALMA Computing received a recommendation to "*confer with human interface specialists*" to "*create more uniform GUIs and GUIs that provide essential information in an intuitive way*".

ALMA Computing responded to the spirit of this recommendation and looked for an active collaboration with Human Computer Interaction (HCI) experts in May 2009. A review of the existing GUIs conducted by the team of experts under the leadership the HLA subsystem group, resulted in a 64 page report submitted to ALMA Computing management in early 2010. The key conclusion was that relatively little attention had been given to the ALMA GUIs, and that several decisions impacting them had been made from a software engineering perspective, ignoring the actual needs of users. The report continued stating that for critical and complex systems, such as ALMA, careful user interface design was

important, and given the complexity of the GUIs, this work needed to be conducted by people knowledgeable in terms of user interface design and well-aware of the state of the art in terms of human computer interaction and information visualization.

The work that followed was a fruitful collaboration with ALMA Computing (not without some initial skepticism from management) that is still going on today. During the years, HCI experts provided not only detailed recommendations on how to improve the existing graphical user interfaces, such as the Observing Tool (OT), Operations Monitoring and Control (OMC), and Quick-Look (QL), which were incorporated in subsequent versions of the ALMA software, but also contributed with actual man power in prototyping some of them. Four participatory design workshops were carried out at the OSF starting late 2010, with the direct participation of key stakeholders (i.e. the actual users of the software), to capture in the field how the tools were being used, and to discuss and prototype additional improvements.

A key concept assimilated during this collaboration was that HCI was not about creating cool looking applications with fancy graphics and animations but about enabling people to do their job better and more efficiently. With this in mind, most of the recommendations implemented followed two main driving principles:

a) minimize memory load on the user, minimize input actions by user; and

b) involve users early in the development of displays and procedures.

What we would have done differently? We would have started HCI earlier, giving more attention to operational user interfaces design during the development, involving end users (operators, astronomers on duty) as soon as possible to capture their perspective, and aiming to reduce operational complexity to enable them to work more efficiently. Participatory design workshops, where white board discussions with stakeholders were quickly translated into draft UI prototypes, have been instrumental to achieve these goals.

3. CONCLUSION

ALMA has been, and still is, an interesting, challenging, sometimes frustrating, but mostly satisfying journey for all of us. Very few have endured the entire adventure, some have moved on to new challenges or finished their professional careers, while others have moved in, bringing new experiences and ideas into the project. While inevitably, working over such a long time on such a complex project, sometimes emotions ran high and intense arguments occurred, one of the most remarkable outcomes of this project is that the spirit of *one* ALMA Computing team delivering *the* ALMA software was never endangered. Mutual respect at all levels and, indeed, long-lasting friendships resulted out of the many years of close collaboration. If this was not the case, we could not have succeeded in compiling such a self-critical reflection on our worldwide collaboration. Nevertheless we want to reiterate that none of the above is laying any blame on any individual or group for any wrongdoing. Quite the opposite, we want to thank all former and current members of the ALMA Computing team for their hard work and dedication to this project.

We hope that this little paper helps you to take the right decision at whichever level for future projects, be it in astronomy or elsewhere. Feel free to contact any of the authors for additional information at any time. If we have helped just to avoid one wrong decision in one project, our time compiling this paper was well spent.

ACKNOWLEDGEMENTS

We would like to thank the following people for providing invaluable input to this paper:

Alan Bridger, United Kingdom Astronomy Technology Centre, Edinburgh, UK

Alisdair Manning, European Southern Observatory, Garching bei München, Germany

Dirk Muders, Max-Planck-Institut für Radioastronomie, Bonn, Germany

Matias Mora, National Radio Astronomy Observatory, Charlottesville, VA, USA

Rafael Hiriart, National Radio Astronomy Observatory, Charlottesville, VA, USA

Tzu-Chiang Shen, Joint ALMA Observatory, Santiago de Chile, Chile

REFERENCES

[1] Schwarz, J. and Raffi, G. “ALMA software architecture”, Proceedings of SPIE Vol. 4848, 55 (2002)

[2] Mora, M., Ibsen, J., Kern, J., Araya, R., Troncoso, N., Gonzalez, V., Marson, R., Juerges, T., Farris, A., and Reyes, C., “Hardware device simulation framework in the ALMA Control subsystem,” in [Astronomical Data Analysis Software and Systems XVIII], Astronomical Society of the Pacific Conference 411 (Sep 2009).

[3] Mora, M., Avarias, J., Tejeda, A., Gil, J.P., and Sommer, H., "ALMA software scalability experience with growing number of antennas," in [SPIE Astronomical Telescopes and Instrumentation], Software and Cyberinfrastructure for Astronomy II 8451, International Society for Optics and Photonics (Jul 2012).

[4] Mora, M., Crew, G., and Rottmann, H., “Phasing up ALMA,” in [These proceedings], (Jun 2014).

[5] Lopez, B., Araya, R., Barriga, N., et al., "Software regression testing: practical experience at the ALMA test facility," Proceedings of SPIE Vol. 7019, 70192X (2008).

[6] Marson, R., Pokorny, M., et. al., “ALMA test interferometer control system: past experiences and future developments,” Proceedings of the SPIE Volume 5496, pp. 12-20 (2004).