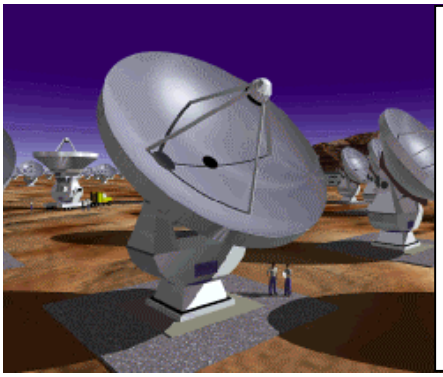

	Atacama Large Millimeter Array	ALMA-SW-0013 Revision: 3 2001-07-202002- *Software Standard* Michele Zamparelli

ALMA Software Engineering Practices

Software Standards

Michele Zamparelli
European Southern Observatory

Keywords: standards, practices, software, procedures

Author Signature:	Date:
Approved by: Brian Glendenning Institute: NRAO	Signature: Date:
Released by: Richard Kurz Institute: ESO	Signature: Date:

Change Record

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
1	29-08-2001	Michele Zamparelli	initial draft
2	18-04-2002	Michele Zamparelli	implementing minutes of review
3	26-11-2002	Michele Zamparelli	all
	Draft for IDR		

Table of Contents

1. Introduction

1.1 Scope

This document covers all the software engineering practices in place for the ALMA software development. It is issued and maintained by the Software Engineering (SE) responsible persons, within the ALMA computing group.

It lists all goals for SE for ALMA, documenting for each goal the choices of procedures and tools to reach the goal plus the verification methodologies to measure the distance to its achievement.

The goals stated in this document represent the basis for agreement between the ALMA Software Management and the SE responsible persons about SE and Quality Assurance activities for ALMA.

At the same time, this document will provide the outline of applicable procedures to be used by the ALMA Software developers (both internal and outsourced) as mandated by the ALMA Software Management.

Only the activities listed in each chapter of this document shall be considered as part of SE for the ALMA computing group. Activities as Risk Management or detailed Planning are outside of this scope and shall be specified, when applicable, at subsystem level in management documents. ~~The explicit use of metrics for design and code evaluation is considered premature at this stage and will be left to later versions of this document.~~

Procedures, methodologies and tools mentioned in this document may be either adopted standards (which must be adhered to) or suggested guidelines (whose adoption is encouraged but not enforced).

A periodic review of this document will guarantee that the goals are in line with the management's view and that their implementation is broadly discussed among the developers' base. Occasionally, guidelines may be promoted to standards and standards may be demoted to guidelines.

The content of this document markedly reflects some specific features of the ALMA software project:

- its geographical and cultural dispersity
- its large foreseen time span
- the twofold type of software involved (hardware control and data processing)
- the scarcity of testing sites for hardware-coupled code

The ALMA SW project has been divided by the ALMA executive committee into two phases: *Phase 1* was used to gather experience in dealing with specific topics of software development (procedures, prototyping, design, software

engineering etc.) whereas *Phase 2* will mark the start of construction both in hardware and software.

1.2 Structure

This document aims to be the central "backbone" document explaining in broad terms all applicable policies. When this is deemed not enough it will reference more detailed documentation. Each chapter covers a specific area where formal Software Practices are considered necessary. Each chapter has:

- a short rationale section containing a brief outline of the practice
- a section specifying the scope
- a section specifying the applicability
- a section specifying the necessary references to procedures and tools, when applicable.

Each chapter is intentionally kept concise to allow the reader to get quickly to the core idea and also providing references to more detailed and technical explanations. Neither the division of the ALMA software system into subsystems, nor the contractual aspects involved with it are addressed in this document.

1.3 Glossary

A Project wide glossary for terms relevant to software is available at http://www.alma.nrao.edu/development/computing/docs/joint/draft/Glossary.htm

The following additional terms are used within this document:

ACG	**ALMA Computing Group**
ADT	**ALMA Development Team, responsible for a Subsystem**
CDR 1-3	**(Incremental) Critical Design Review 1,2,3**
FTE	**Full Time Equivalent**
ICD	**Interface Control Document**
Package	**Major component of Subsystem**
Project Computing	**Task or Subsystem activity of ALMA**
PAR	**Preliminary Acceptance Review**
PDR	**Preliminary Design Review**
R0-5	**Release of software (and its number)**

Subsystem Subsystem of the ALMA Software System

1.4 References

1. http://www.testing.com/ , B. Marick
2. UML Distilled, second edition, M. Fowler, S. Kendall, 1999 Addison-Wesley
3. ALMA-SW-NNNN, 1.1, 2002~~1~~-11~~07~~-10~~23~~, ALMA Computing Plan for Phase2, G. Raffi
4. ALMA-SW-NNNN,2,2001-05-15, ALMA Computing Plan for Phase 1, B. Glendenning, G. Raffi
5. The Common Object Request Broker Architecture & Specification, 2.4.2, 2001-02-01, OMG

~~6.ALMA-PRO-ESO-xxxxx-xxxx,1,2000-07-28, Software Devlopment Process: Methodology and Tools, G.Chiozzi, R.Karban. P.Sivera~~

~~7.~~6. ALMA-SW-0015, 1, ALMA Software Document Review Procedure, M. Zamparelli

~~8.~~7. VLT-MAN-PECS for Users, Administrators, Application Developers and PECS Developers, 1, 30-11-00. A. Huxley

~~9.~~8. ALMA-SW-0008,5, 2001-06-06, ALMA Software Documentation Standard, A.Bridger

~~10.~~9. ALMA-SW-0012, 1, ALMA Software and Hardware Standards, M. Zamparelli

~~11.~~10. VLT-MAN-ESO-17200-0780, 1.3, 2001-04-23, Configuration Management Module User Manual

~~12.~~11. ALMA-SW-0009, 3, 2001-06-06, C Coding Standards, A. Bridger, M. Brooks, J. Pisano

~~13.~~12. ALMA-SW-0010, 4, 2001-06-11, C++ Coding Standards, A. Bridger, J.Pisano

~~14.~~13. ESO Action Remedy System: http://support.eso.org/ars/cgi-bin/arweb

~~15.~~14. http://www.omg.org/gettingstarted/omg_idl.htm

~~16.~~15. Linux Installation manual TOBESPECIFIED

~~17.~~16. AcsMakefile manpages

~~18.~~17. http://www.alma.nrao.edu/development/computing/docs/index.html

~~19.~~**~~18.~~** Code Complete, Steve McConnel [1993]; Microsoft Press.~~TOBESPECIFIED~~

19. CMM to CVS Transition Plan, ALMA-SW-XXXX, M.Zamparelli (Technical Memo)

20. Agile Developmen in ALMA, Markus Völter (Technical Memo)

21. ALMAEDM, http://almaedm.tuc.nrao.edu/

22. ALMA Software Engineering webpage. http://www.eso.org/projects/alma/develop/alma-se/reference/IntegrationTools.html

2. Software Process

2.1 Goal

Several distinct tasks are necessary to deliver a software product, and a methodology is needed to know who should be doing what, when and how in order to reach the target.

Every software development group requires a reference scheme to achieve an organized, controlled way of working. A framework specifying tasks, timelines, methods, deliverables and exchange formats guides the developers towards their goal, and helps in pulling the group together.

~~To achieve this, a process inspired to the Unified Software Development Process (UP), currently an industry mainstream methodology, will be used.~~

This process, ~~defined in [6],~~ divides the development effort into several phases, to be carried out following an iterative improvement scheme. ~~The UP~~ It is based on the Unified Modeling Language (UML) which provides a standard way to visualize, specify, construct, document and communicate the artifacts of a software system design and architecture (more on UML can be found in [2]).

The software life cycle ~~as defined in the UP~~ is divided into 4 Phases (the first iteration of Inception is done only once). The end of each phase is marked by a milestone, when a certain set of artifacts are made available by the project developers: some are new, some are artifacts from previous milestones which are enriched with a deeper level of detail or a broader coverage. For a list of artifacts see Appendix A. The milestones ~~are properly addressed in [6], and~~ can be summarized in the following table:

Phase	***Milestone/ Document Name***	***Purpose***
First Inception	*Software Requirements*	**Understand the basic requirements of the system** to be developed.
Inception	*Preliminary Design*	• **Define system scope**, i.e. it has to identify what is inside and what is outside the system. The basic interfaces between the system and the Actors are sketched and they fit with what is provided or foreseen for the Actors. • **Identify an architecture** that can implement the requirements expressed for the system. The high level internal structure of the system is defined in a realistic way. • **Identify and mitigate** the risks critical to the successful implementation of the system. Risks can come from many technical and non-technical areas.

Elaboration	*Critical Design*	Identify a robust and resilient architecture baseline Identify and mitigate major risks Support with a proper project plan a realist estimate of schedule, cost and quality. The Critical Design Review must demonstrate that these objectives have been reached and the stakeholders officially accept the proposed architecture. On the other side, the organization responsible for the development finally commits itself in delivering the product with the agreed features and within the agreed budget and planning forecasts. This milestone is the no-return point in the project and all major risks must have been investigated.
Construction	*Preliminary Acceptance*	~~actually build the system~~Construction of the system.. The System Delivery milestone marks the end of the Construction Phase and shall demonstrate that the system has reached a level of product capability suitable for initial operation in the final environment. In the first iteration, the system still contains bugs and imperfections, but can be used.
Transition	*Commissioning*	make the system ready for unrestricted release to the development or user community.

It has to be pointed out that not all of these milestones need to correspond to resource-intensive formal reviews, in most cases an update of previous artifacts will suffice. Notice also that the basic activities (*core workflows*) of requirement capturing, analysis, design, implementation and testing are found, with different emphasis, in each phase.

No methodology is mandated as to how to get from the deliverables of one phase those of the next.

The end of a cycle is marked by a *release*. The number of releases per year planned for the ALMA software system is specified in [3]. Each release will have a running number or label identifying it, in the form :

Major.Minor.Patch

This numbering scheme is meant to easily identify whether a release represents a major change (interface or design document change), a minor change (extension of functionality according to planned design) or a patch due to the necessity to correct a software error.

Prior to each release, a particular person or group will be assigned the responsibility for the producing the release. This will entail certification of

the testing (see [11]) and review of the documentation to assess its quality and sufficiency. More on the release procedure can be found in chapter 10.

2.2 Scope

All software subsystems in ALMA.

2.3 Applicability

Phase 1 and Phase 2.

2.4 Procedures and Tools

To engineer a model of the system, ALMA software designers are entitled to use any drawing tool or UML tool. The use of ~~will use~~ the tool *Rational Rose* (~~. It will be m~~made available at all ALMA development sites) is encouraged since it provides the broadest range of model verification possibilities. Any similar tool generating artifacts in a mainstream format (XMI) is accepted.

~~The document: [6] is currently in preparation for submission as draft and describes a set of tools to facilitate the creation and handling of printed and online documentation necessary for the various milestones.~~

3. Document Reviews

3.1 Goal

Explicit agreement in certain technical areas or on some procedural issues is required before programming activities start in earnest.

Therefore a system of formal reviews for ALMA software documents will be set up and applied systematically. This will force the software team to reach agreement before development occurs. The purpose is also to achieve a sufficient level of discussion for all those topics, which are relevant enough to the software development or maintenance.

Though large review boards usually imply long discussions or hot debates, they also ensure that key information is properly disseminated to a significant sample of the concerned parties.

Updates to existing documents do not need to be formally reviewed if the changes are minor (for example, formatting) and the SPR system described in chapter 8 can be used to keep track of these changes. Unless they contain proprietary or otherwise confidential information, all documents are to be made publicly available.

3.2 Scope

All Computing Memos (see 4), released by the ALMA computing group for internal or external usage are subject to formal review processes specified in [6~~7~~] and made available online.

All documents, which may be given to outside companies for contractual ends will have to be signed by the appropriate authority.

3.3 Applicability

Phase 1 and Phase 2

3.4 Procedures and Tools

A simple tool can be engineered, which sorts comments according to originator and page number in order to make collection easier.

The document review procedure is contained in [6~~7~~].

4. Document Formats, Templates and Numbering Scheme

4.1 Goal

All documents issued by the ALMA Computing group must follow similar layout and stylistic conventions. It must be possible and easy to identify and exchange documents among the various ALMA sites and institutions.

The ALMA Software Management has adopted Microsoft Word as the format for its documentation. Other tools capable to truly interoperate with this format may also be used. The choice, while controversial, is motivated by the large availability of the product, the wealth of third-party add-ons and its expected long term availability on the market.

The ALMA Software Management has adopted a general layout for all documents concerning software or software activities. The layout is accompanied by a template for Microsoft Word.

As specified in [8~~9~~], the document will have a label of the form:

ALMA-SW-NNNN

where NNNN is a running number provided by the ALMA Software Management. Most software documents will be sorted into one of the following main categories:

- **Computing Memos**: documents that have been approved by all concerned members of the ALMA project's Computing Division. While anyone may submit a general ALMA memo, only memos that follow the review procedures described in [3] will be considered to represent the technical consensus or policy of the group

- **Technical Memo**: informal technical reports with limited scope, which are not likely to change in time.

In addition to this, several kinds of documents are produced during the software life cycle and are bound to a specific subsystem release: User Manuals, Installation Manuals, Interface Control Documents, Test Procedures; most of them will be part of the deliverables for the various milestones at subsystem level.

The interfaces of subsystems will be represented by Application Programming Interface (API) documents. These documents will contain information generated from heavily annotated source code interface files (e.g. *.h or *.java files) and are not supposed to follow the standards specified in [8~~9~~] nor the review of [6~~7~~]. Instead, they are meant to be periodically regenerated whenever the source code changes. Tools have been purchased which roughly measure the sufficiency of such documentation for C++ and Java programming languages.

With the exception of the code extracted documentation, all documents will comply to the standard defined in [8~~9~~] and will be either made available ~~on~~

~~the web (isee Appendix C) or inn the a~~ designated ~~yet to be selected~~ document management system (21). In either case previous releases of each document plus the minutes of the review meetings will be available.

4.2 Scope

All documents created by the ALMA Computing Group.

4.3 Applicability

Phase 2

4.4 Procedures and Tools

The ALMA Software documentation standard is specified in [8~~9~~].

~~There is currently no~~The officially defined ALMA ~~or ALMA SW~~ documentation ~~or product documentation~~ archive for storage, and retrieval can be found in (21) and has the following features:~~. Such a system will have to:~~

- ~~allow~~ reliable access from geographically dispersed locations
- ~~be~~ web based with minimum amount of client side dedicated code
- allow document workflow, with e-mail messaging to author, reviewers and management for each document state change
- allow grouping of documentation in areas protected by suitable access control lists and authentication mechanism.

~~Such a system is currently being selected among various industry standard candidates.~~ The product Forum from PTC has been recently selected. ~~In the meantime~~ ~~d~~Documents archived so far have been ~~are~~ stored in [17~~18~~], where a librarian ensures they are in the correct format, adds the minutes of the reviews, and ensures they are up-to-date.

The document extraction tool currently in use is *doxygen*.

5. Interface Control

5.1 Goal

Interface Control documents specify the interchange between two or more software systems. Due to their importance, they must be subject to special control, in order to avoid arbitrary change by responsible of individual software packages or subsystems.

Interface documents will be specified using IDL (Interface Definition Language) whenever possible and will be initially defined following a formal review process as specified in [6~~7~~]. Other information as data formats or reference to external tools may also be needed.

Subsequent changes will be discussed using the SPR system explained in chapter 11.

Interfaces between hardware and software subsystems will be administered directly by the ALMA System Engineering Group.

5.2 Scope

All interfaces for all software modules developed in parallel.

5.3 Applicability

Phase 2

5.4 Procedures and Tools

The IDL specifications can be found in [5] or in [14~~15~~].

6. Development Environment

6.1 Goal

The software/hardware environment used by ALMA software developers must be as similar and homogeneous as possible. This is crucial to the project's success all the more if one considers ALMA's geographical dispersion. It concerns not only operating systems and tools in the developer's workbench, also generic middle layer or third party software are involved.

To achieve this goal, the usage of the ALMA Common Software (ACS~~), of the Pluggable Environment Contribution System (PECS))~~ and of the acsMakefile is mandatory.

The ACS has been designed to provide the communication layer between the majority of user applications needed for the ALMA software system. Applications utilizing the ACS are guaranteed to have a common messaging, event notification and logging system, just to mention a few.

Furthermore, ACS provides:

1. detailed OS installation manual, covering step by step an installation from scratch, up to partition layout, kernel tuning etc. etc.

2. software toolbox needed for development, with compilers, editors, debuggers. A single project compiler for instance, is needed in order to ensure maximum build stability.

3. the acsMakefile, an extension of the GNU makefile allowing a clear separation between system wide rules and settings and the specifics of a module's build procedure.

~~4.PECS, a complex but extremely versatile system to harmonize two or more applications and their required environment variables at default, system, and user level. It allows for instance the addition of a new application system wide, by simply adding files to a directory, without need of modification of system wide files.~~

It has to be pointed out that though delivered together, the ACS library and the set of tools and workbench for software development are two separate entities.

The Software Engineering group of ALMA Computing will gradually take over from ACS the responsibility of choosing and maintaining the abovementioned tools and OS.

6.2 Scope

The ACS covers software tools running mostly on Unix systems and VxWorks, the only exception being IDEs for Java, which run on Windows.

6.3 Applicability

Phase 1 & Phase 2

6.4 Procedures and Tools

The ACS comes with the so-called ACS Standard Environment. This influences developers' activities in a variety of ways, but most notably through the following environment variables:

$MODROOT	the development area where you are now working
$INTROOT	the integration area currently in use
$ACSROOT	the ACS SW root currently in use

The hierarchy in using files is: development area, integration area, system areaand is obtained using the following search paths:

commands	$MODROOT/bin	$INTROOT/bin	$ACSROOT/bin
include files	$MODROOT/include	$INTROOT/include	$ACSROOT/include
libxxx.a:	$MODROOT/lib	$INTROOT/lib	$ACSROOT/lib
CLASSPATH	$MODROOT/lib	$INTROOT/lib	$ACSROOT/lib

The ACS also provides standard templates for directories, makefiles and C++ header files.

The development tools contained in the ACS toolbox are listed in the release notes for each release.

The ALMA Software Management will decide the OS release the ACS will have to be ported to. It is anticipated that the ACS will lag behind the latest release for a specific OS, until the latter has achieved the desired amount of stability. In a similar way, the latest version of any tool will not be adopted by the ACS until it has reached sufficient guarantees of stability in the software community. OS installation manuals for ACS can be found in [15~~16~~]

The ALMA Software Management shall be notified should some requested functionality not be available in the ACS.

More info on the acsMakefile can be found in [16~~17~~].

~~A detailed description of PECS and its raison d'être can be found in: [8].~~

7. Integration Environment

7.1 Goal

An integration of all modules, packages and subsystem composing the ALMA software system will have to be carried out periodically. It is necessary that this task be automated as much as possible.

Each institution contributing to the ALMA Software effort will have integration responsibility for the subsystem, package or module it produces. This means to allocate enough manpower and hardware resources to this end (dedicated integration machines). The Integration and Test group will assume ~~opportunity and the extent of a~~ centralized integration responsibility, and promote ~~or at least of a~~ strong coordination among subsystems in integration issues.

~~will have to be assessed during Phase 2.~~

It's important to realize that the integration for the purely data processing part and for the hardware-coupled control system follow different guidelines.

The integrators will carry out following steps:

~~□verification that all development code compiles and links successfully (anything additional for CORBA????)~~ I1	~~□~~verification that all development code compiles and links successfully
~~□~~I2 ~~verification of the adoption of approved coding standards (see chapter~~	verification of the adoption of approved coding standards (see chapter 8)

8) □	
□I3analysis of test coverage for each module, using a suitable tool.	□exercise unit testing to verify whether the test suite specified by the developer are executed successfully.
□I4 exercise unit testing to verify whether the test suite specified by the developer are executed successfully.	□analysis of test coverage for each module, using a suitable tool.
□I5run–time checking against memory leaks and boundary violations using suitable tools.	□ run–time checking against memory leaks and boundary violations using suitable tools.

The integration shall be carried on a monthly basis. Every first of the month releases by the various subsystems will have to be made available to the Integration and Test group. This will happen *in addition* to the release schedule outlined in The frequency and depth of the integration process must be chosen by the ALMA Software Management and should respect the importance or critical role of the various subsystems. Regional competence centers for the integration of the various subsystems will be established.

7.2 Scope

All software subsystems

7.3 Applicability

Phase 2

7.4 Procedures and Tools

A tool to be used for checking memory leaks will be made available for developers at all sites. An up-to-date table summarizing tools adoption for the various development and integration phases can be found in: 22 Sites with specific integration responsabilities will use *Rational Purify* for C++ and *Sitraka Jprobe* for Java. The majority of tools has still to be evaluated and selected. Notice that subsystems are entitled to deviate from the suggestions of Software Engineering, but the latter will only have sufficient manpower to support one tool for each category/language/operating system.

8. Coding Standards

8.1 Goal

ALMA software must be readable, easy to maintain and as least error prone as possible. It will have to use official standards as much as possible.

To this purpose, coding standards will be applied systematically and a list of general computing standards will be compiled and made available in [9~~10~~].

Coding standards range from proper code documentation to file naming conventions and in general help in preventing a certain category of software bugs. They do not address specific implementation details like algorithms or programming methodologies.

The application of coding standards is deemed crucial to any modern software undertaking.

8.2 Scope

Coding standards have been approved and are applicable for the languages C and C++ and may be found in [11~~12~~] and [12~~13~~]. ~~The opportunity to mandate~~ Coding standards for other languages are planned as well (IDL, Java, Python)~~. is currently open to debate.~~

8.3 Applicability:

Phase2

8.4 Procedures and Tools

Applicable standards are defined in: (11) and (12).

S~~A s~~uitable tools to verify their adoption has ~~to be~~been selected and are currently being phased in (Codewizard and JTest). Such ~~a~~ tools will be made available both to software developers and integrators, it will have to be executable in batch mode and provide clearly readable reports. The tools will be first used in a centralized way as Quality Control, and then distributed to the SDT as appropriate. ~~Possibilities range between all-out commercial packages and GNU based public domain tools whose configuration would have to be maintained in house.~~

General computing standards will be described in: (9)

currently in preparation.

9. Code Inspections

9.1 Goal

Adoption of approved coding standards must be periodically monitored and the results of this monitoring process must be made available to developers and management.

This can be achieved by inspections of the code ("code copping"). Both manual (human) and automatic inspections are possible, though it's important to recognize the limitations and the costs of both. Source code will be subject to periodic scrutiny (at package level) by suitable software tools (see 8.4) which will rate the code according to compliance to predefined guidelines.

Note that code inspection is not meant primarily to detect faults, but merely to check the programming style.

Furthermore, to achieve a certain homogeneity within the ALMA Computing group, to promote similar techniques and methodologies and facilitate re-tasking of human resources, the ALMA SW board will promote periodic cross inspections of code by developers. Inspections will not be carried out for each release nor for each software subsystem.

Management may decide that human inspections are mandatory for certain packages of special relevance or for those code segments which exhibit a remarkably high algorithmic complexity.

9.2 Scope:

Inspections will not cover the design or the interface parts, but will be applied mostly to source code or to testing code.

9.3 Applicability :

Phase 2

9.4 Procedures and Tools

Coding standards/Best practices adoption tools are mentioned in ~~See~~ 8.4. Also, tools measuring McCabe cyclomatic complexity have been purchased and are being phased in. This measure, together with other industry standard metrics is going to be used merely as a maintenance tool, to determine for instance which software modules are more prone to exhibit faulty behaviour, and should therefore be tested more thoroughly.

10.Configuration Management

10.1 Goal

Software produced by the ALMA Computing group must be stored in a centralized archive and accessible by each group member. It must be possible to track and identify previous system configurations.

A Configuration Management tool must be adopted to this end.

Configuration Management "is the discipline of identifying the configuration of a system at discrete points in time for purposes of systematically controlling changes to this configuration and maintaining the integrity and traceability of this configuration throughout the system life cycle". [1]

Software developers use a Configuration Management tool to baseline configurations, prepare releases and to deal with synchronous update of items. For release preparation in particular, code freezing or "tagging" is required i.e. a simple notification by the subsystem responsible, that a certain version of the corresponding module has achieved its planned objectives and may be taken from the repository for the release.

The Configuration Management tool for ALMA must enable:

- identification of items

~~strict locking mechanism~~

- traceability of changes (who did what, when)
- accessibility of previous configurations for any item

10.2 Scope

Both source code and documentation will be subject to strict version control, though in principle, two different software tools may be used.

10.3 Applicability

Phase 1 and Phase 2

10.4 Procedures and Tools

Developers using Configuration Management will adopt the following minimal set of rules:

[1] Technical Report CMU/SEI-99-TR-004

- Each module shall have one responsible person, who will be entitled to delegate activities on the module but shall retain responsibility at all times.
- items should be locked for the minimum period of time consistent with completing the work required
- source code should be checked in after verification that it does not break the build
- descriptive, non-empty comments are required for each file/module on check in. They must be complete and readable, making reference to SPR calls (see section 12) when applicable, avoiding “standard” phraseology like “fixed typo” or “fixed bug”

~~□~~The system chose for Phase 2 is CVS (Concurrent Version system) a public domain tool which has been available for many years and offers sufficient guarantees of reliability and stability. ~~comments will be entered both at module level in the Changelog file, and at file level in the header part.~~

~~As of this writing the system under evaluation is CMM with~~ The ~~a~~ source ~~and documentation~~ repository has been set up at the ESO facilities. ~~It requires a one-sentence comment when checking in a module.~~

Such repository will have to guarantee a high level of fault tolerance and availability. Duplication of repositories to be used in case of network outage is currently not foreseen. ~~The current system will undergo a review in accordance to plan [4].~~

11.Testing

11.1 Goal

The amount of software faults or incorrect behaviours in the ALMA software system must be kept to a minimum and the system must be validated, i.e. it must be guaranteed that it is working according to its specifications, prior to Final Acceptance Test.

The application of a consistent testing scheme and the diffusion of a "testing culture" will help to achieve this goal.

Although the developer is encouraged to delegate test code writing to someone else, it is his/her final responsibility to make sure that his/her module has achieved a sufficient degree of testing. Scripting languages are deemed practical for writing test code since they attain the necessary simplicity, but in general no specific language is mandated.

A formal testing scheme will be adopted to ensure developers check-in only modules, which have been previously tested. During integration (see section 7), software subsystems may be rejected if they do not provide sufficient testing certification.

Developers are required to start working on their test suites as a result of design, prior to implementation. The responsible for each software subsystem will make sure that two types of regression tests are performed:

- Unit tests: the smallest compilable unit is tested under isolation. If needed, the behaviour of other code units interacting with the unit under test will be mimicked by building stubs.

- System tests: the system (or subsystem) as a whole is tested against its functional specifications

For exact definitions of regression, unit, black box and functional tests see [1] and [18~~19~~].

11.2 Scope

All source code. Currently, no executable UML models will be used, so no design will be "tested".

11.3 Applicability

Phase 2

11.4 Procedures and Tools

Agreement will have to be reached on:

- language to use

- directory names for test code, test code names for unit and functional tests
- testing environment to be used, giving the developers the freedom to express all initial conditions and testing requirements they might have

The configuration specification and the testing code will be used by appropriate integration groups for integration activities.

Additional testing guidelines will be provided either in terms of a separate ALMA document or as reference to well established testing literature.

In order to determine the amount of code coverage of each test suite and thus its sensibility an appropriate tool will be distributed both to developers and integration groups. There will have to be a tool for Unix and one for the real time OS.

The TAT tool, which comes along with the ACS shall be used as an interim solution.

12. Change Management

12.1 Goal

Keeping track of faults in software products whenever these are discovered is of fundamental importance.

To this end, a Software Problem Report (SPR) has been put in place for the ALMA Computing Group. It is based on Action Remedy, located and maintained via a Web interface at the ESO premises. The SPR system asks its user to provide the application field plus all configuration management specific information necessary for developers to reproduce the incorrect behaviour or fault. The latter includes the specification of the concerned software subsystem.

An ALMA SPR Board will convene periodically to address, prioritize and task all **new** calls. At the end of each such meeting, every new SPR will have been either closed (as mistaken or already solved) or assigned to somebody to be implemented for a certain release or milestone.

Is it suggested to follow a zero based planning for handling the SPRs: following each major release an additional ALMA SPR Board will convene to assess the situation of all those calls which are still **in progress**, to rediscuss their priority and importance.

The SPR system is meant to be used to raise the attention on issues of limited scope with immediate impact on daily work. All other issues, like long term strategy changes are outside of its scope.

The workflow for an SPR call is briefly: open, in progress or closed. Each closed SPR has an additional label (closed status) specifying if it was: solved (a solution has been put in place), clarified (a misunderstanding or dispute about documentation) or rejected (the SPR has no basis to exist). A notification is sent by e-mail to the SPR originator plus all those listed as Carbon Copy for the package, whenever a status change takes place.

The SPR system constitutes the main mechanism for feedback from the users community during ALMA's operational lifetime.

12.2 Scope

Software and documentation

12.3 Applicability

Phase 1 and Phase 2

12.4 Procedures and Tools

The SPR system is available in: [13~~14~~].

12.5

Appendix A. Deliverables for the Software Process

This Appendix assumes familiarity with UML and the Use-Case concepts. Notice that in general the deliverables for one phase also contain the deliverables of the previous phase in a refined, up-to-date or more detailed version.

A.1. Software Requirements Review

Deliverables:

Glossary and overall system description. A general description of the system to be developed and a glossary with the definition of the terms used. They must create a common background between all stakeholders and team members and avoid misunderstanding. If the domain of the project is not well known domain analysis should be done and a domain model produced.

System Context Diagram. The context diagram shows the system under design as a black box and all the Actors that interact with the system.

Actors. A description of all the entities interacting with the system. Primary Actors are the users of the system (not necessarily human, also other software systems). Secondary Actors are all the sub-systems that are part of the system and that have to be controlled by software or that are needed to fulfill the requirements (like a Time Reference System, that is necessary to satisfy time precision requirements).

Use Case Model. All the Use Cases are derived from the user requirements and from higher level documents**.** The development team is responsible for writing the Use Cases based on whatever information is initially provided by the stakeholders. When Use Cases have been inferred from the available initial documentation they are discussed with the stakeholders and accordingly modified. The language used to write the Use Cases is simple enough for the stakeholders to understand it, but the process to obtain them requires technical knowledge that only the development team can have.

When use cases have not been written, textual requirements information in form of numbered lists will be provided.

General and non-functional requirements These requirements specify the adoption of specific standards, hardware architectures, software libraries or system performances, maintainability, extensibility and reliability. Some of these requirements fit in the bodies of the Use Cases, but for most of them specific document sections have to be written, imposed by project standards and by contour constraints.

Risk assessment. A first basic analysis of the areas of major risks in the project.

A.2. Design Preliminary Review

Purpose

Preliminary Design is aimed at defining the system scope, identify the architecture and the risks involved in the system.

The Preliminary Design Review has to demonstrate that all these objectives have been reached and that it is feasible for the team to proceed with the project, i.e. the team has the technical capabilities to implement the proposed architecture.

A successful Preliminary Design Review also demonstrates that the stakeholders agree on the requirements identified and on the objectives stated. They give their agreement to proceed with the next phase.

Deliverables:

All deliverables of the previous phase must be reviewed and detailed.

Package Documentation consisting of:

- **Package Description.** A textual description of the package, following a predefined template
- **Package Class Diagram.** A first Class Diagram where every package is just represented by a class. It allows representing the basic relations between the packages
- **Package Use Case Diagram.** It shows all Use Cases that are responsibility of the package and the relations with Actors.

Interfaces. There must be one ICD sub-section per Actor. Every ICD is subdivided in Interfaces, where every Interface describes a small number of Operations that are highly internally coherent and loosely coupled with other interfaces.

The ICD sections are used to extract the ICD documents for not already implemented/existing Actors and to check the already implemented interfaces with the existing Actors.

It has to be ensured that all the needed Interfaces are specified. As soon as more detailed information are available the ICDs are updated.

No IDL files are expected for preliminary design.

Deployment Diagram and Process view. The Deployment Diagram is a Structural Diagram. It shows a set of nodes and their relationships. This diagram is used to show the static deployment view of the architecture, i.e. the allocation of processes, as identified in the packages, to the processing nodes in the physical design of the system. It is essential to be able to perform a performance analysis.

Performance Analysis. It is essential for PDR to demonstrate that a reasonable estimation of the performances required is available and that they can be met with the available HW. This has been done through a table with processes/packages allocated to HW:

- CPU power/budget
- Estimated CPU consumption of SW, based on experience or previous projects

Prototypes. If applicable, a prototype of user interface will be provided

Testing Specifications: Definition of test environments and simulation conditions for software to be integrated during the various releases.

Design of critical Use Cases: For the most important and/or complex Use Cases it is necessary to provide more details, and in particular it is necessary to demonstrate how the proposed system design is going to allow the implementation of the Use Cases.

For this purpose some behavioral diagrams can be used:

- **Activity diagrams.** An activity diagram shows the flow from activity to activity within a system (and in particular for what concerns a specific Use Case). The diagram is especially important in modeling the function of a system and emphasizes the flow of control among sub-systems or, going to higher design details, objects.

- **Interaction (Collaboration and Sequence) Diagrams.** These diagrams are essential to model the dynamic aspects of a system. An interaction diagram shows an interaction, consisting of a set of objects (at our design level, sub-systems) and their relationships, including the messages that may be dispatched among them. The Sequence Diagram emphasizes the time ordering of messages. The Collaboration Diagram emphasizes the structural organization of the objects that send and receive messages.

- **State Diagrams.** State Diagrams show a state machine, consisting of states, transitions, events and activities. They are particularly important in modeling the behavior of an interface, class and collaboration. They emphasize the event-ordered behavior of an object, which is especially useful in modeling reactive systems.

Though these diagrams are suggested, this list is not exhaustive and other types of interaction diagrams may be used to complement or as alternative.

Planning for the PDR documentation kit a planning has to be provided, showing planned dates for Milestones, description of the purpose of each milestone plus deliverables. The planning activity is described in more detail in a separate managerial document outlining the specifics of subsystems development.

An essential part of the planning activity consists also in assigning a priority level to all Use Cases or eventually to Use Cases sub-flows, i.e. to scenarios. The content of a release and of iteration in the Construction Phase is actually determined by identifying which Use Cases (or scenarios) will have to be implemented, based on their priority.

This will be formalized in the planning delivered for CDR, that has to contain a mapping of all Use Cases associated to any release, to allow formal tracing.

The System Design Description document contains also ALL information that is part of the Requirements Specification. Readers interested only in requirements

should get this last document. Readers interested also in more details and in the system architecture need to get only the System Design Description.

A.3. Critical Design Review

Deliverables

During the whole Elaboration Phase the team works on the items already delivered for the Preliminary Design Review, adding new details.

It is very important in this phase to develop prototypes to verify the proposed architecture and to analyze the major risks. The prototypes also help in estimating the time and resources necessary for the implementation, in particular when no historical data coming from previous projects are available.

Building prototypes as complete as possible in terms of control electronics and including a number of hardware components is considered very critical. These prototypes are used in the elaboration phase to assess the proposed architecture and verify that critical requirements can be satisfied.

During this phase, most of the time is used in the Analysis and Design work-flows to build the architecture of the packages

The basic analysis classes of a package are obtained from the step by step analysis of all the Use Cases under the responsibility of the package itself.

These analysis classes always fit in one of the following basic three categories:

Boundary classes A boundary class is used to model the interaction between the package and the Actors, i.e. an exchange of information or of action requests between the package and the Actors or other packages in the system.A change in an interface is usually isolated in one or more boundary classes.In every packages there is typically one boundary class per every Actor interacting with it. It implements all interactions identified in the Use Cases assigned to the package.

Entity classes An entity class holds information that typically lasts beyond the life of a Use Case. Entity classes are identified by:

- Finding from each Use Case description the information-bearing objects
- Finding them from the problem domain
- Finding them from the original requirement documents

Only classes needed in some Use Case must be introduced. One has to begin the search in the Use Cases and use the other sources to confirm the choice and structure of the classes identified.

Control classes A control class represents coordination, sequencing, transactions and control of other objects. The dynamics of the system are modeled by control classes. There is typically a control class per each Use Case, although simple Use Cases may not need a control class.

A.4. Preliminary Acceptance Review

Purpose:

In the Construction Phase, the emphasis shifts from the accumulation of the knowledge necessary to build the system to its actual construction

Deliverables:

Executable System: the set of binary executable files for the appropriate Operating System as extracted and build from the repository, when released.

List of implemented Use Cases for each Package.

Test Cases for each Package. Each package will be tested independently. Black box testing is based on Use Cases. Scenarios are used to produce Test Cases. Stubs replace external packages. Open box test is based on Class Tests. Test Cases for system integration are obtained from the Use Cases.

The prototype developed for the Elaboration phase becomes now a "Control Model" and used for modular testing of subsystem packages (that need to have access to specific electronic or hardware components) and for integration testing. Code is organized in modules where each contains a regression test that can be run in an autonomous and automatic way.

Component diagrams are created, which derive from the package and class diagrams and map directly to implementation units.

If detailed design affects the system and high level design the corresponding documentation has to be updated.

The code documentation is integrated in the online documentation to have a complete reference available. In particular they are linked to the component diagrams.

A.5. Final Acceptance Test

The only area that needs changes to be properly integrated in the Use Case driven process is the definition of the Test Cases. Test Cases for final acceptance are directly extracted from the Use Cases. This is needed in order to meet the objective of tracing requirements through the whole process down to final acceptance test by using Use Cases. All test procedures must be fully automatic or, when this is not possible, based on a detailed checklist.

Purpose:

The purpose of the Transition Phase is to make the system ready for unrestricted release to the user community.

Deliverables:

External release of final system.

Acceptance Test procedure reports

Final user and maintenance documentation. Use Cases can be extremely valuable for writing maintenance and user documentation

Appendix B. List of supported Third-Party Software Tools

Type	Name	Licence Info	Point of Contact	Status
Word Processor	Microsoft Word 97	OEM	local	in use
Software Modeling Tool	Rational Rose	alma-sw-semgr@eso.org	local	in use
Configuration Management	CMM	inernal ESO	ESO	in use
Software documentation extractor	Doxygen	GPL	local	in use
Fault Reporting System	Action Remedy	remedy@eso.org	ESO	in use

Appendix C. Information Technology Support

The following mailing lists have been put in place for the ALMA Computing Group. Searchable email archives of all lists will be made available on the web.

Listname	**Meaning**
alma-sw-workers@nrao.edu	everybody active in software development
alma-sw-announce@nrao.edu	everybody interested in new draft documents available for review
alma-sw-practices@nrao.edu	everybody concerned with Software Engineering Practices
alma-sw-common@nrao.edu	everybody concerned with ALMA Common Software

The following WWW Addresses are used by the ALMA Computing Group:

	drafts repository
http://www.alma.nrao.edu/development/computing/docs/memos/index.html	reviewed documents repository