


Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		


## LOFAR Software Engineering Guideline

Verified:			
Name	Signature	Date	Rev.nr.
Marco de Vos		2002/10/17	1.0

Accepted:		
	System Engineering Manager	Program Manager
	Marco de Vos	Jan Reitsma

<p>© ASTRON 2002</p> <p>All rights are reserved. Reproduction in whole or in part is prohibited without written consent of the copyright owner.</p>
---




Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

## Distribution list:

ISC	WPMs (for Project Team)	TAC
Harvey Butcher (ASTRON) Joe Salah (MIT) Phil Schwarz (NRL)	ASTRON: Jaap Bregman Jan Noordam Kjeld van der Schaaf Dion Kant Wim van Cappellen Jan Doornink (DutchSpace) Hans Kollen (DutchSpace) Martijn van Veelen Ger van Diepen Frank van Eck (Ordina) Hans de Wolf (DutchSpace) Albert-Jan Boonstra Rob Millenaar Willem Baan	Wim Brouw (CSIRO) Bill Cotton (NRAO) Tom Clark Steve Ellingson (Ohio State University) Alle-Jan van der Veen (Delft Technical University)
PMT		SCB
Jan Reitsma (ASTRON) Colin Lonsdale (MIT/Haystack) Davidson Chen (NRL)		Namir Kassim (NRL, chair) Michiel van Haarlem (ASTRON) Ger de Bruyn (Groningen University/ASTRON) <i>Huub Rottgering (Leiden University)</i> Bryan Gaensler (Harvard University) Rob Fender (University of Amsterdam) Colin Lonsdale (MIT/Haystack) Jim Cordes (Cornell University) Frazer Owen (NRAO)
SEG		
Roger Cappallo (MIT/Haystack) Marco de Vos (ASTRON) Lee J. Rickard (NRL)	MIT/Haystack: Jackie Hewitt Shep Doeleman Brian Corey Jody Attridge Chris Phillips  NRL: Bill Erickson Namir Kassim Kurt Weiler Joe Lazio Ken Stewart Brian Hicks Pat Crane Paul Ray	


## Document history:

Revision	Date	Section	Page(s)	Modification
0.5	2000-10-16		-	Creation

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

## Table of contents:

1	Introduction .....	4
1.1	Purpose of this document .....	4
1.2	Executive summary.....	4
2	Configuration Management.....	5
3	Development Approach.....	9
3.1	Iterative or waterfall?.....	9
3.2	Discussion of popular software development methods .....	11
3.3	Prince2.....	14
3.4	The LOFAR approach .....	15
3.5	Limitations to Subsystem development methodology.....	16
4	The software Engineering Life cycle .....	17
4.1	Requirements.....	17
4.2	Architecture and detailed design .....	17
4.3	Modelling .....	17
4.4	Operation System .....	17
4.5	Middleware .....	17
4.6	Languages, code standard.....	17
4.7	Tooling .....	18
4.8	Testing en verification .....	18
4.9	Reviews .....	18
	References.....	19

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

# 1 Introduction

## 1.1 Purpose of this document

This document provides a guideline for software development within the LOFAR project. The context for this guideline document is given by the LOFAR project standards; the software engineering approach should be an integral part of the LOFAR project management.

This document briefly describes the Configuration Management approach for the LOFAR project as a whole, and software development activities in particular. The configuration management for software activities will be embedded in the LOFAR project wide configuration management system, to which some artefacts are added.


The software development process approach is discussed next. The software development approach must correspond to the LOFAR system engineering approach. Especially with respect to system integration the two engineering disciplines must co-operate closely.

Finally, guidelines for all steps within the software development life cycle are given. These guidelines embody the more general approaches given in the preceding sections. This section contains the body of guidelines to be used by members of the software development teams.

The software development guidelines presented in this document are of course completely applicable to those subsystems concerned with pure software developments, such as MAC, SAS, LCS and CPA. Part of the guidelines is also applicable to digital signal processing systems, embedded software or HW/SW systems. Such systems are found in the SDP and CEP Platform subsystems.

## 1.2 Executive summary

[TBW]

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

## 2 Configuration Management

Configuration management is an important aspect of software development projects. Especially version control and (automated) build management are important operational tools during the engineering workflow. The software configuration management must be strongly embedded in the LOFAR project as a whole. Major parts of the software engineering management are equally relevant for hardware development and system engineering. The differences may be found in detailed and dynamically evolving version numbering and (automated) regular builds and regression testing. For these aspects some dedicated information logging may be required in addition to the “normal” bookkeeping.

In the current status of the LOFAR project, no final configuration management approach exists yet, although an information database system is defined and used for the requirements management. Therefore, in the current version of this document a complete configuration management approach is discussed, large parts of which may be replaced by references to appropriate LOFAR documents later on.


The configuration management approach is based on the information model shown in Figure 1. The information model is shown as a class diagram in which the classes have stereotype <<table>>. The model can be implemented as a relational database with one table per class. The relations between the classes can be used for tracking and tracing. The information model is discussed in more detail in [11].

The implementation of the CM information model into a working CM system can be performed gradually during the project. In the current phase of the project (before PDR), the tracking of requirement and specification changes is less important, so a simple implementation of this part is feasible. Also, the build and verification tables are not heavily used. On the other hand, this is a good time to start developing and using the CM system.

The implementation of the CM system should provide dedicated GUI access to sets of relevant tables for typical usage. Those GUIs also provide knowledge of how to relate the tables and possibly how to access version control, file systems etc. The following GUIs should be provided:

- Build process
- Regression testing
- Problem report/ change request handling
- Issue and action management
- [TBW: others]

The workflow of problem report and enhancement requests management, including change request boards etc., will be discussed in the Configuration Management approach (see [11] and [7]).

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

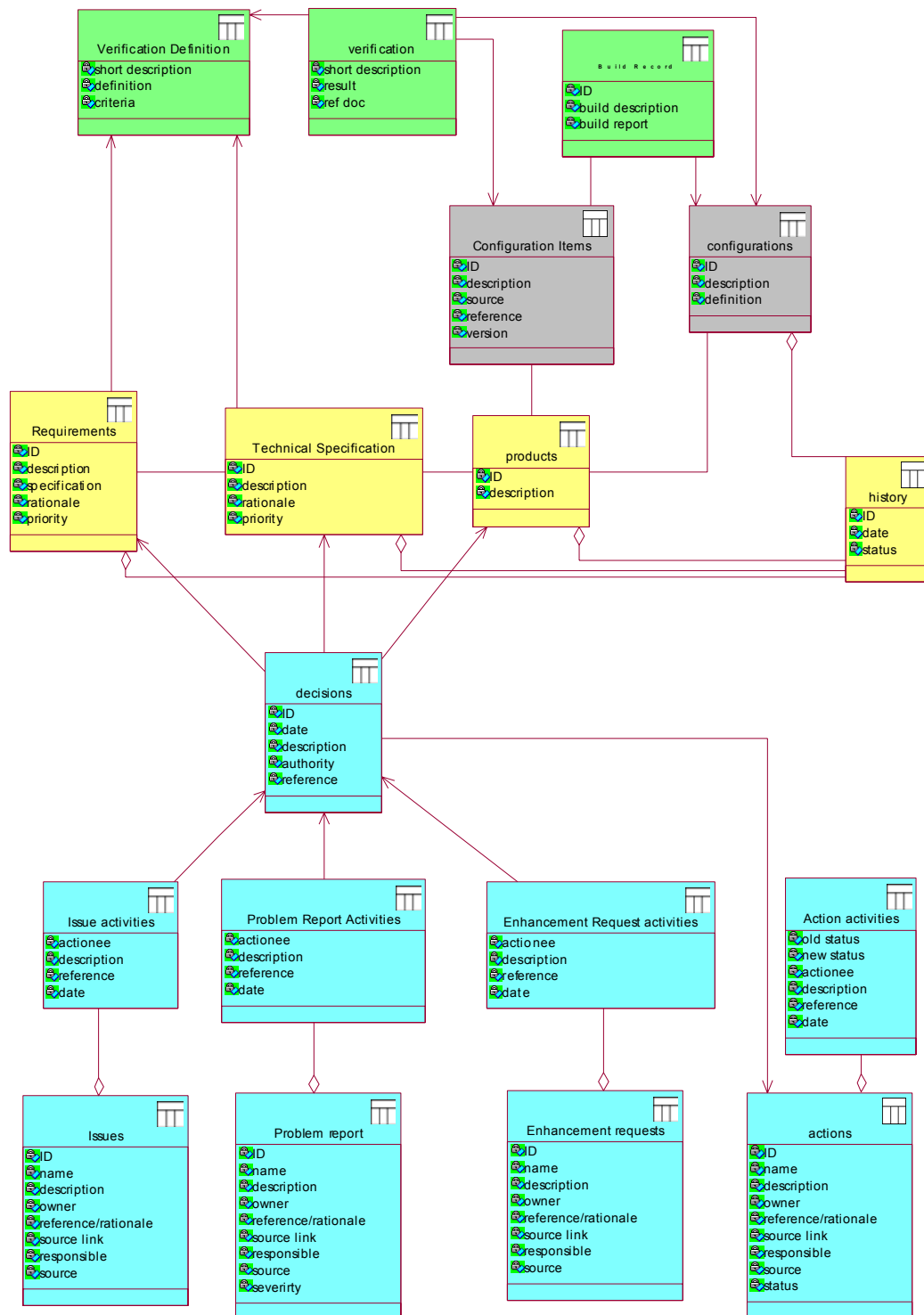




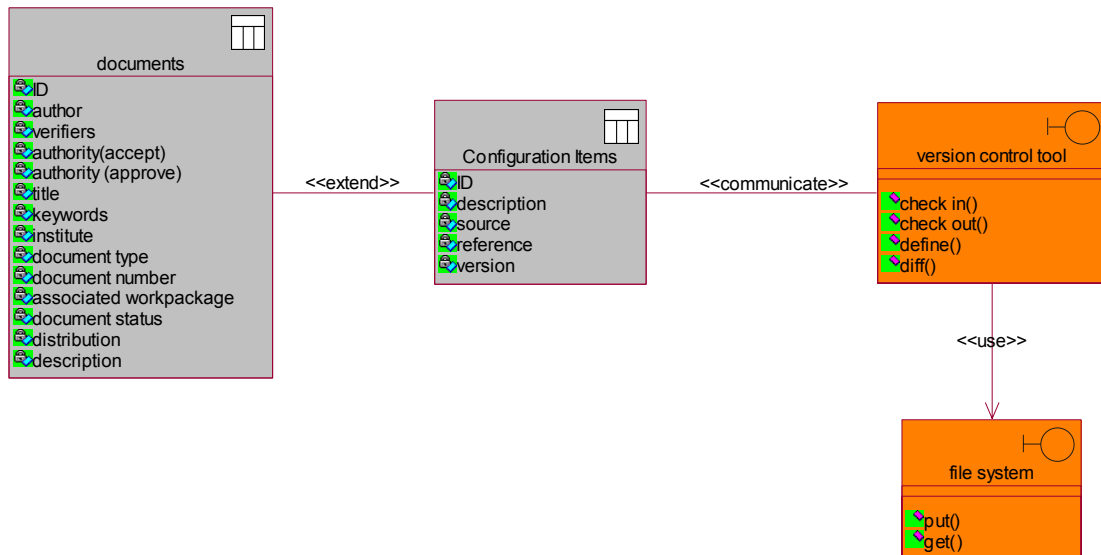
Figure 1 Configuration Management information model.

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

The most important tables and their functions are:

- **Requirements.** This table contains the requirements and specifications of the LOFAR system and subsystems. The testability is defined in the verification table; a relation to this table is given for the verification definition and for the performed verifications.  
Associated with the requirements is a requirement history table. The requirement history entries may define links to decisions.  
The Requirements are ordered according to their appearance in SRS documents through the *Requirements Sections* table (not shown in Figure 1).
- **Technical Specifications.** This table defines the technical specifications of the LOFAR system and its subsystems. The specifications are ordered according to their appearance in ADD documents through the *Specification Sections* table (not shown in Figure 1). Associated with the specifications is a requirement history table. The specification history entries may define links to decisions.
- **Products.** The *Products* table covers a hierarchy of products. The hierarchical relations are defined in this table. The number of levels in such a hierarchy may depend on the product itself. The product hierarchical relations (parent child relations) are defined in a *Product Hierarchy* table.  
Products can be classes of CIs. Therefore, the CI table can be seen as instantiations of products.
- **Configuration Description.** There are two major types of configurations:
  - An assembly of products in a certain configuration (setup). E.g. the definition of a lab setup consisting of a product, test equipment, connections etc.
  - A configuration description describes the build (production) process of actual CIs. For example, a configuration may describe the production process for the product "LF antenna version 3.1". In this example, the configuration description is used in build "LF antenna batch 45 on 23 February 2003" resulting in CIs for 100 antennas. Another example is the build of a software library on a particular computer configuration.
- **Build Record.** The build record defines and describes a build. A build is the transformation of one or more configuration items into a new configuration item. For example, two modules of source files are compiled and linked producing an executable. The two modules each are a configuration item, as is the executable. In the build record the build processes is described (build definition) and the results are logged.
- **Verification and Verification definition.** These tables contain records for all verification definitions and activities. A verification definition describes and defines a verification process. The definition contains the exact script of how to perform a verification. Criteria for pass/fail are defined. Each requirement or specification should be associated with one or more verification definitions that define when the requirement or specification is fulfilled.  
A verification activity relates to one or more configuration items that are verified using a configuration. The verification is defined in a verification definition record. For example, an executable program (configuration item) is run on a cluster computer (configuration) and crashes during execution (verification result).  
The short description field gives an informal description of the verification. The exact definition of the verification process, for example a test script, is given in the definition field.
- **History.** The history of evolving items is traced through the history table. Such information is administered for the following tables: Requirements, Specifications, Products and configuration. The status field is used to indicate the currently valid item.
- **Issues, Actions and Changes.** These tables are all specializations of topics. For each topic an activity table is kept. In this activity table the activities related to the topic are administered. Issues, problem reports and enhancement requests lead to decisions. A possible decision is to perform a new action. Also, the decision may imply modifications to the requirement, specification and product tables. This is the only valid way to modify those tables (apart from initial data entry).  
It is possible to promote an Issue into an action or problem report/enhancement request through a decision (this should be a one-button action in the actual implementation of the CM system).

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		




**Figure 2** The relation of configuration items to the version control tool and the relation between configuration items and documents. Documents are configuration items for which extra attributes are defined in a documents table class.

The relation of version control and documents to the configuration item table is shown in Figure 2. A document is treated as an extension to the configuration item table.

A version control tool is used to keep track of modifications to files. This is relevant for all files on which the developer has influence and that determine the configuration items. For software development, these are in first place the source files. However, also model definitions (e.g. rose model files) and other design definitions should be controlled. Also, the files controlling the build process (e.g. autoconf macros), IDE settings and settings for simulator programs should be controlled. Results of a build process must not be put under version control since these are on-off actions; those can be stored on a normal file system and referred to from the build record table.

The version field in the Configuration Items table class defines how the actual configuration items can be retrieved from the configuration management tool. For example, a *tag* can be specified for a CVS based version control system.



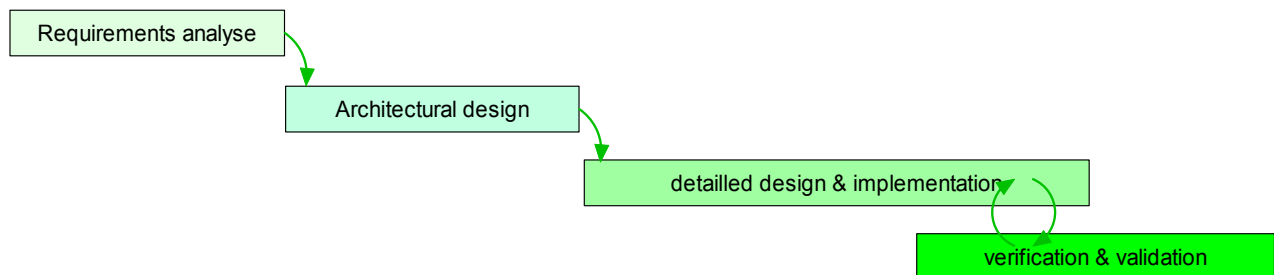
Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

### 3 Development Approach

This section discusses the software development (life cycle) approach. The LOFAR system level software development (and system integration!) approach is described. Within this context, subsystems software development may use different approaches, but the system level context does provide boundary conditions.

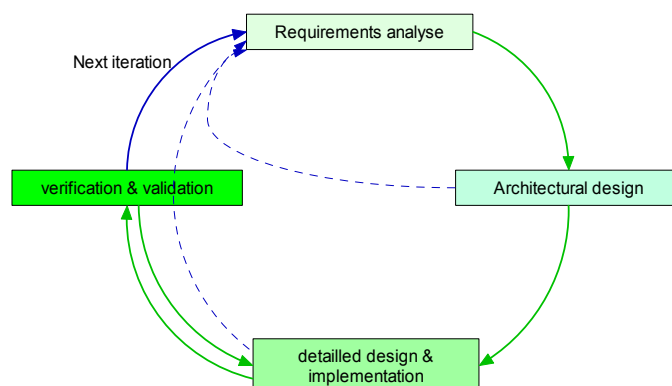
#### 3.1 Iterative or waterfall?

We first start with a short definition of the terms waterfall and iterative development. In Figure 3 the waterfall development approach is shown. The whole development is performed step by step. First the complete requirements analysis is performed. Next the complete architectural design is made after which the detailed design and implementation is done. The final verification may find some bugs that are to be solved in the implementation block, yielding some interaction between the final two stages.




**Figure 3** Waterfall or revolutionary process. The process evolves in one direction, without feedback between the stages. Only during the verification process, some feedback to the implementation process is needed (bugs!).

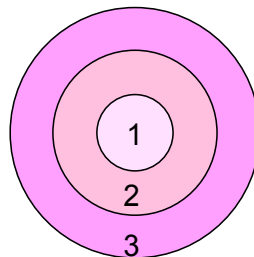
The iterative process approach is illustrated in Figure 4. The processing stages are similar to those in the waterfall approach. However, this time we go through the stages multiple times. In each iteration, all stages are performed, this is sometimes called a “mini project”. The results of an iteration is input for the next iteration.



**Figure 4** Iterative process. The process steps are executed in multiple iterations. Each iteration contains all steps and uses feedback from former iterations.

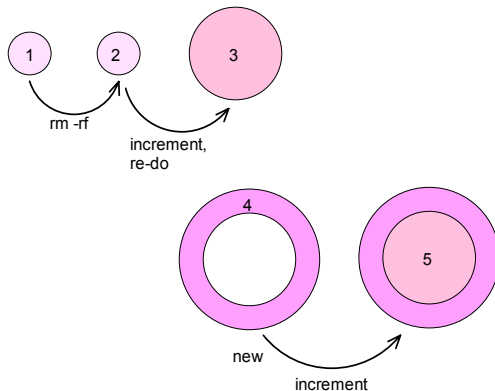
Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

In iterative development, we can use multiple models of how to work towards the desired end product. In Figure 5 we see an example of how multiple iterations contribute to the final software product. The final product is developed by adding more and more functionality in each iteration, this is called incremental development.




**Figure 5** "Onion" style iterative development; every iteration extends the performance of the system.

On the other hand, we can also use one or more iterations to test a principle, architecture or so. Such a test may generate knowledge that can be used in later iterations, but may not be part of the final deliverable. Also, an iteration may (partly) re-do some of the work done in a previous iteration. These possibilities are shown in Figure 6.



**Figure 6** Other types of iteration results: No increment at all (1-2), complete --better-- reprogramming (2-3), new module (4) and integration of modules (3-5)

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

## 3.2 Discussion of popular software development methods

In the next sub sections, the particular features of the three most popular iterative software development approaches are discussed. The discussion focuses on the usage of (parts of) the methods for the LOFAR software development approach. A prerequisite is that we will not adopt a single complete method, but rather will provide a general description based on a mixture of approaches, leaving open the possibility to adapt a particular approach for the development of a particular subsystem, for example for development outsourced at third parties.

In the current version of this document, this section is at the discussion level. In later versions more definite choices will have been made and indications may be given of how to incorporate the given approach in the three described standard methods.

### 3.2.1 Unified Process


*People are involved in the development of a software product throughout its entire life cycle. They finance the product, schedule it, develop it, manage it, test it, use it, and benefit from it. Therefore, the process that guides this development must be people oriented, that is, one that works well for the people using it.*  
--- Jacobson et al. [6]

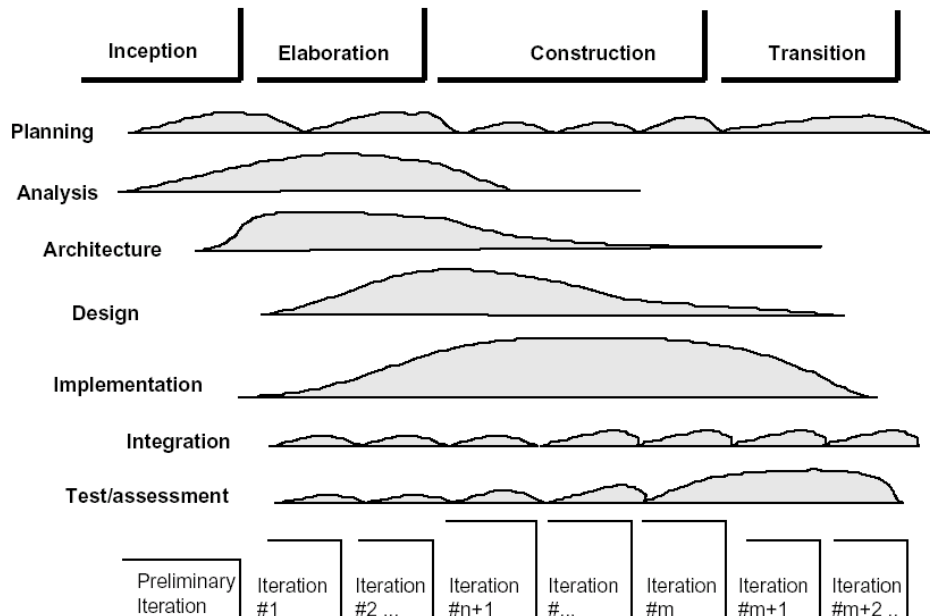
The unified process describes how the overall waterfall-like project stages are translated into a large set of short iterations in the software development process [2]. During the project, the focus of the iterations changes from requirements finding, analysis and architecture definition to implementations and testing. This is described by the project phases: Inception – Elaboration – Construction – Transition. All iterations do follow the same waterfall-like core workflows: planning - analysis – architecture – design – implementation – integration - test.

The unified process is use-case driven. Use cases are not just a tool for specifying the requirements of the system; they also drive the system's design, implementation and test. That is, use cases drive the complete development process. Based on the use-case model, developers create a series of design and implementation models that realise the use cases. The developers review the models for conformance to the use-cases. The testers verify the software components for compliance to the original use-cases.

[TBW: Unified process is an architecture driven process]

[TBW: roles and workers]

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		



**Figure 7** The core workflows --- planning, analysis, architecture, design, implementation, integration and test --- take place over the four phases: inception, elaboration, construction and transition.


[TBW: pros and cons of (R)UP]

### 3.2.2 DSDM

*DSDM is more a framework than a method. It does not say how things should be done in details, but provides a skeleton process and product description that are to be tailored to suit a particular project or a particular organisation.*  
--- DSDM Manual

The project process, as shown in Figure 8, has five phases: Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration and finally Implementation in the working environment:

- **Feasibility Study.** This phase includes a definition of the problem to be addressed together with assessments of the likely costs and of the technical feasibility of delivering a system to solve the business problem.
- **Business Study.** This study produces a global overview of the system to be build. Also, the system architecture definition is produced, which is the basis for the functional modelling in the next iteration.
- **Functional Model Iteration.** The focus of Functional Model Iteration is on refining the business-based aspects of the system, i.e. building on the high-level processing and information requirements identified during the Business Study. Both the Functional Model Iteration and the Design and Build Iteration consist of cycles of four activities: Identify what is to be produced, agree how and when to do it, create the product and finally check that it has been produced correctly. The bulk of development work is in the two iteration phases where prototypes are incrementally built towards the tested system. All prototypes in DSDM are intended to evolve into the final system and are therefore built to be robust enough for operational use and to satisfy any relevant non-functional requirements, such as performance.

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

- **Design and Build Iteration.** The Design and Build Iteration is where the system is engineered to a sufficiently high standard to be safely placed in the hands of the users. The major product here is the Tested System. The DSDM process diagram does not show testing as a distinct activity because testing is happening throughout both the Functional Model Iteration and the Design and Build Iteration. Some environments or contractual arrangements will require separate testing phases to be included at the end of the development of the increment, but this should not be the major activity encountered in more traditional approaches to development. Testing is just as important in DSDM and consumes just as much effort, but it is spread throughout development.
- **Implementation.** The Implementation phase covers the migration from the development environment to the operational environment. This includes training the users who have not been part of the project team. One product of this phase is the Increment Review Document. The Increment Review Document is used to summarise what the project has achieved in terms of its short-term objectives.




**Figure 8** *The DSDM process model.*

Important techniques used in DSDM are timeboxing and the MoSCoW rules for requirements prioritisation:

- **M**ust Haves fundamental to the projects success
- **S**hould Haves important but the projects success does not rely on these
- **C**ould Haves can easily be left out without impacting on the project
- **W**on't Have this time round can be left out this time and done at a later date

DSDM and Unified Process can be combined. In this case DSDM is used for business driven rapid application development in the construction phase of a more rigorous architecture driven Unified Process approach.

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

### 3.2.3 EVO

*One of the great time-wasters in software projects is detailed requirements analysis, followed by detailed design, followed by full coding and testing phases. If only we had the intellectual capacity, and the necessary knowledge, to do those thing accurately! In reality, we have to admit that we cannot tackle such tasks adequately for any but trivially small projects. There are too many unknowns, too many dynamic changes , and too complex a set of interrelationships in the systems we build. We must take a more humble approach.*  
--- Tom Gilb [4]

Similar to DSDM, in EVO every iteration in the software development process focuses on maximal contribution to the user requirements. Each iteration does deliver value to the end-user. Also, the set of user requirements to be fulfilled in the next iteration is based on the (user!) experiences with the current deliverable. Early, frequent iterations are used to get a maximal involvement of the user in the project.

An open-ended basic system architecture is favoured. An open-ended architecture is characterised by maximal adaptability, extendibility, portability and improvability. Such an architecture is needed in order to add the new components corresponding to new or modified requirements found in later iterations of the development process.

[TBW: more information on EVO development]


[TBW: pros and cons]

### 3.3 Prince2

Prince2 is a process management method and not a development method. It is often used for the management of software projects, possibly in combination with one of the development methods discussed. Prince2 may be used for (software) subsystem development. The Prince2 method must be tailored to fit the DDV context, but the similarity between the method and the ECCS-E-10A standard should allow for this.

[TBW: short analysis of the differences]

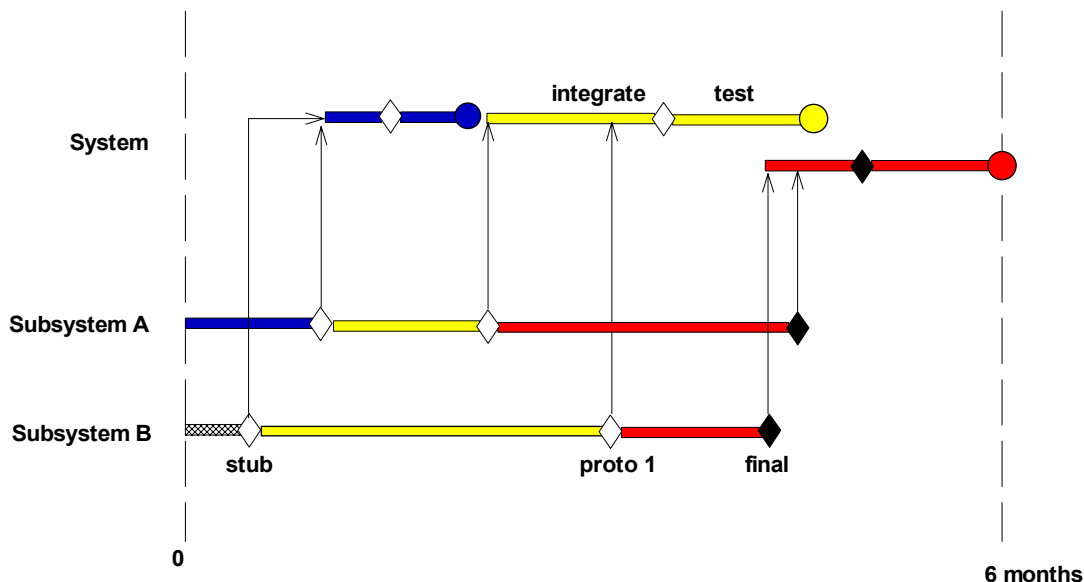
[TBW: short analysis of the pros and cons of prince2 for LOFAR subsystems]

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

### 3.4 The LOFAR approach

A modular approach to the software development will be followed. The total functionality to be provided by software is divided in a rather large set of modules with low coupling between modules and high in-module cohesion. This subdivision is provided at system level by software system engineers. The delivery scheme of those modules is based on the planned integration scheme of prototypes and final products. This scheme requires multiple (partly) prototypes for each subsystem. For modules within the subsystems, either multiple prototypes are requested (thus implying an iterative development process) or stubs may be provided instead.


In Figure 9 the life cycle approaches for subsystem deliveries is shown. At system level, a life cycle frequency of half a year is chosen. Within this half-year, proto types and stubs from the subsystems are continuously integrated and tested. The system level integration team consists of a small permanent staff to which developers from the subsystem teams are added for short periods during the integration of subsystem deliverables.



**Figure 9** System level integration and subsystem delivery life cycles.

The LOFAR top-level development process is described in the LOFAR Template [8], which is based on the ECSS-E-10A standard. This DDV plan provides the context for the software subsystems development. The subsystems themselves can be developed using existing software development methods appropriate for the dynamics and context for the subsystem. In general, iterative methods should be used in order to provide early proto types for the system level integration activities. Unified process and DSDM are favoured. Existing practices at subcontractors should be an important aspect in the choice for a specific method.

[TBW]


Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

### 3.5 Limitations to Subsystem development methodology.

The LOFAR software development approach defines the context for subsystem software development. The subsystems may adopt one of the software development methods described before. However some of the assumptions taking in the various methods may be hard to fulfil:

- **Unified Process:** [TBW]
- **DSDM:** A high degree of involvement of the “user” is required. For subsystem development, part of the user role should be provided by LOFAR system level engineers. The dynamics of the LOFAR project determines the availability of those engineers to a large extend. Therefore, the user role may not always be available, which may conflict with the high interaction required for the DSDM method. DSDM does not provide a rigorous architecture-driven software development approach.
- **EVO:** [TBW]



Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

## 4 The software Engineering Life cycle

This section describes details of the software development lifecycle. This gives a guideline of how aspects in the software engineering workflow must be done.

In the current version of this document only a raw overview is given.

### 4.1 Requirements

[TBW: Use case gathering and definition]

[TBW: from use case to requirements and specs]

[TBW: requirements management; see CM]

### 4.2 Architecture and detailed design

[TBW: Architecture guideline]

[TBW: patterns etc.]

### 4.3 Modelling

The Unified Modelling Language (UML) is used for modelling of software products. "Structured" data flow diagrams should not be used since they try to tell too much. Instead, activity diagrams are used to define the logical/functional level. Class and Object (or collaboration) diagrams are used to define the data transport. The preferred UML case tool is Rational Rose.

[TBW]

### 4.4 Operation System

Unix operating systems should be used, unless there are good arguments not to do so. The preferred Unix variant is Linux.

### 4.5 Middleware

Middleware libraries shall be used where possible.

Preferred libraries are: MPICH, Corba, AIPS++ [TBD]

[TBW: more libraries]

[TBW: what to use when]

[TBW: standards and implementations]

### 4.6 Languages, code standard


C++ is the preferred programming language. Other languages are used for:

- GUI: [TBD, e.g. Java, Qt] Device drivers: ANSI C Scripting: To be defined in LCS subsystem.

A set of coding standards shall be provided covering all languages. Sets of languages may use a generic coding standard, for example C, C++ and Java will have a large common set of coding rules. Currently a C++ coding standard is used in the CEP workpackage [10]. This standard will be split in a generic part and a C++ specific section. These two documents will form the basis for the LOFAR coding standard documents. Dedicated coding standard documents must be made for drivers and GUIs.

The coding standard documents shall also address detailed design issues.

All coding standard documents will contain a 1-page (at most) shortlist of most used rules, which can be used during daily development work.

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

Common software components are administered (and probably also maintained) by the LCS workpackage. Those components address “generic” issues that are expected to be useful for multiple software products. There are two important motivations for re-use of such generic software components:

- Standardisation enhances understandability, maintenance etc.
- Re-use can reduce development time.

The current list of common software components (from the CEP and SIM workpackages) is:

- Debug and tracing macros
- Mutex style locking mechanisms
- [TBW]

## 4.7 Tooling

**CASE:** Rational Rose

**Code documentation:** Doxygen. The C++ code standard defines how code documentation shall be marked.

**Static Code checking:** [TBD]

**Dynamic Code checking:** Insure++.

**C++ Compiler:** gnu 2.95 (will be phased out during project), Gnu 3.X will be the basis for software development. Production code may use optimised and/or hardware specific compilers ([TBD]).

**IDE:** eclipse ?

**Build environment:** Based on Autotools, including regression testing based on scripts. A working environment is developed in the CEP workpackage, see [ref]. This will be the basis for the LOFAR software build environment.

## 4.8 Testing en verification

[TBW: V-model]

[TBW: Code inspection.]

[TBW: Code reviews.]

[TBW: Requirements review.]

[TBW: Architecture review.]

[TBW: Module testing is the basis test]


[TBW: Hardware built-in tests for embedded systems]

Regression testing is part of the build environment. Each software module in the build environment has an associated set of test programmes and test scripts. The tests script defines how to execute the test programmes and defines the input and output files. The generated output file is compared with a pre-defined one, yielding the test result.

## 4.9 Reviews

[TBW: when to review]

[TBW: how to review]

Author: Kjeld v.d. Schaaf	Date of issue: 2002-10-16 Kind of issue: limited	Scope: project/sde Doc.id: LOFAR-ASTRON-MEM-060	
	Status: draft Revision nr: 0.5		

## References

- [1] F. Buschmann *et al*, *A System of patterns*, Wiley, 1996, ISBN 0471958697
- [2] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley 1999
- [3] E. Gamma *et al.*, *Design Patterns*, Addison Wesley 1994
- [4] T. Gilb, *Principles of Software Engineering Management*, Addison Wesley 1988, isbn 0201192462
- [5] C. Hoffman, R. Nord and D. Soni, *Applied Software Architecture*, Addison Wesley 2000
- [6] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, 1999
- [7] H.Kollen, *LOFAR Configuration Management Plan*, LOFAR-ASTRON-PLN-008
- [8] H.Kollen, *LOFAR Design, Development & Verification Plan*, LOFAR-ASTRON-PLN-007, 2002
- [9] P. Kruchten, *Architectural Blueprints – The “4+1” View Model of Software Architecture*, published in IEEE Software 12 (6), November 1995, pp. 42-50
- [10] K. v.d. Schaaf, *LOFAR CEP Software development Coding Standard*, LOFAR-ASTRON-MEM-008
- [11] K.v.d. Schaaf, *LOFAR Configuration Management*, LOFAR-ASTRON-PLN-006
- [12] J. Stapleton, *Dynamic Systems Development Method*, Academic Service 1999, isbn 9039510911 (dutch translation)