PROJECT WORK PLAN

Organization: CERN – LCG Project

SEAL
Common Core Libraries and Services for LHC Applications

Document Revision #: 1.0

Date of Issue: 9.01.2003

Project Manager: Pere Mato

Approval Signatures

Approved by: Project Leader	Approved by: LCG Project Leader
Prepared by: Project Manager	Prepared by: LCG Project Manager
	Reviewed by: Quality Assurance Manager

Table of Contents

Document Change Control

This section provides control for the development and distribution of revisions to the Project Work Plan up to the point of approval.

Revision Number	Date of Issue	Author(s)	Brief Description of Change
0.1	6.01.2003	P. Mato	Initial Draft
0.2	7.01.2003	P. Mato	Changes from Jacek. Completed sections 4 & 5
1.0	9.01.2003	P. Mato	Included changes and suggestions from Stefan, Massimo and Jacek

1. Project Overview

The common core libraries and services (SEAL) project is initiated from the recommendation of the Blueprint RTAG [1] to provide the basic software infrastructure for other LCG application area projects and in common for the LHC experiments.

1.1 Purpose, Scope, and Objectives

The purpose of the project is to provide the software infrastructure, basic frameworks, libraries and tools that are common among the LHC experiments. The project should address the selection, integration, development and support of foundation and utility class libraries. These utilities cover a broad range of unrelated functionalities and it is essentially impossible to find a unique optimum provider for all of them. They should be developed or adapted as the need arises. In addition to these foundation and utility libraries, the project should develop a coherent set of basic framework services to facilitate the integration of LCG and non-LCG software to build coherent applications.

The scope of the SEAL project is basically the scope of the LCG Applications Area. The Applications Area expected scope includes common applications software infrastructure, frameworks, libraries, and tools; common applications such as simulation and analysis toolkits; and assisting and supporting the integration and adaptation of physics applications software in the Grid environment

The SEAL project should provide a **coherent** and as complete as possible set of core classes and services in conformance with overall architectural vision described in the Blueprint RTAG.

The two main deliverables of the project are: *Foundation Class Libraries* and *Basic Framework services*.

Foundation Class Libraries

- Basic Types. These libraries are low level fairly independent class libraries to complement the standard basic types (int, float …). In addition to the basic types which are applicable to a wide range of applications there will be basic types specific to HEP (e.g. LorentzVector).
- Utility libraries. These utilities cover a broad range of unrelated functionalities which makes sense to re-use across LCG projects. These will be developed or adapted as the need arises.
- System libraries. These are the libraries that should provide an operating system isolation layer. Instead of each LCG project or experiment developing its own interface to the system services a common one should be provided.

- Domain specific foundation libraries. Many other core libraries will be needed to implement high level services and software components in many domains (simulation, interactive analysis, etc.). The project should provide support for such libraries.

Basic Framework Services

A coherent, integrated set of core infrastructure and core services supporting the development of higher level framework components and specializations should be addressed by the project. An initial set of basic services have been identified in the Blueprint RTAG.

- Component model. Develop or adapt a basic set of mechanisms and base classes for managing creation of objects (factories), lifetime, multiplicity and scope, component communication and interface discovery.
- Reflection. Reflection is the ability to query a class about its internal structure at run time. This finds applications in object streamers, object browsers, rapid prototyping, etc. Since C++ does not provide this kind of functionality, the project should develop an object dictionary to provide reflection functionality by complementing the native C++ language features.
- Plugging management. In the LCG architecture a plug-in is a logical module that encapsulates the concrete implementation of a given service. The plugging manager is responsible for loading, activating, deactivating and unloading plug-ins at run-time.
- Other framework services. Develop basic framework services for message reporting, exception handling, component configuration, "event" management, object "white board", etc. Other services candidate to be developed will be identified by the other LCG applications area projects.
- Scripting. Provide the basic infrastructure to support scripting. In particular, bindings for Python and CINT of the basic services will be needed to provide a "component bus" that allow easy integration of components providing a wide variety of functionality, possibly implemented in a variety of languages.
- Distributed computing, Grid. Provide a common interface to the Grid services to be used by the other LCG application area projects.

The SEAL project provides core software services and support for foundations libraries to the other LCG application area software projects such as the object persistency project (POOL), math libraries project, physicists interface project (PI). It is expected that the LHC experiment frameworks, which cover additional areas not in the scope of the LCG application area such as event processing and algorithm scheduling framework, will also benefit from the SEAL deliverables by facilitating their integration with the LCG provided functionality. Other HEP software projects not specifically for LHC, with perhaps a wider domain of applicability, can also take advantage of the basic framework services that will be provided by SEAL.

It is expected that new needs and requirements will be uncovered later during the execution of the project. It is also expected that software components or utility libraries originally developed for a specific project will be migrated to the SEAL project as soon as they became candidates for more general re-use.

1.2 Assumptions, Constraints and Risks

The SEAL project is based on the following assumptions:

- Most of the core software which is going to be delivered in a coherent packaging by the project exists in one form or another in the experiment's core software. It is assumed that most of the work will be in re-packaging existing pieces of software, since the resources available are not sufficient for a complete development of the whole of the core software.
- The LHC experiments are committed in using the LCG software when available. It is essential for the success of the project that they provide fast feedback of the deliverables.

These are the known constrains and risks of the project:

- The project will deliver the components as soon they are available. The priorities will be set by the users (other LCG projects and LHC experiments).
- The quality of the core software delivered by the project must be superior to any other software component or application based on it.
- We will re-use as much as possible existing software. This will require making compromises and adaptations to achieve the necessary level of coherency and conformance to the architectural vision already established.
- Any project adopting the software developed by this project will need to be adapted and will be required to replace its own software elements with the ones functionally equivalent provided by the project. This will certainly imply some period of instability for the experiment applications.

1.3 Project Deliverables and Schedule

As already mentioned, the project will deliver the software incrementally. The first releases will fulfill the needs of the other LCG projects. The schedule of the delivery of the first version is summarized as follows:

- Release *V1 alpha* defined as essential functionality sufficient for the other existing LCG projects by end March 2003. Frequent internal releases before this date.
- Release *V1 beta* defined as essential functionality sufficient to be adopted by experiments frameworks by end June 2003

The main milestones of the project are:

- 2002/11/30 Define the V1 SEAL software
- 2002/12/1 Prototype object dictionary service
- 2003/1/15 Establish external software decision process. Establish the process and policies by which decisions are made on what external software is to be used by the LCG applications area.
- 2003/1/31 Complete the initial SEAL work plan. Complete the initial SEAL work plan for submission to the SC2. Should cover (at least) the content and implementation plan for SEAL V1.
- 2003/3/31 SEAL V1 essentials in alpha. The most essential elements of the V1 SEAL suite (as requested by projects needing to use them) are available in alpha.
- 2003/5/31 Grid enabled services defined. The SEAL services which must be grid-enabled are defined and their implementation prioritized.

1.4 Budget Summary

The computing infrastructure required to run the project, not including the developer's desktops, such as test servers, build servers, code repositories, web site are assumed to be provided by the SPI project together with the CERN IT and EP divisions. The staff joining the SEAL project will keep their desktop hardware, which is generally provided by their institute.

1.5 Evolution of the Plan

The structure of this Project Plan is in compliance with the recommendations of IEEE Std 1058-1998.

- The project plan should be revised every six months in order to reflect the changes in requirements and priorities, and the evolution in the level of staffing. A good moment to revise this plan would be just after the release of the of v1 beta during summer 2003.
- As soon as the SC2 committee agrees to this planning, it will be used as a baseline for project tracking. The plan will be controlled and tracked quarterly.
- The updates will be presented to the SC2 together with the tracking of the existing plan.

1.6 References

[1] Report of the LHC Computing Grid Project Architecture Blueprint RTAG, CERN-LCG-2002-022, (http://cern.ch/lcg/SC2/RTAG8/finalreport.doc)

[2] Thoughts on Software Process - V1, T. Wenaus, (http://wenaus.home.cern.ch/wenaus/peb-app/processV1.html)

1.7 Definitions and Acronyms

LCG

LHC Computing Grip project (http://cern.ch/lcg)

SC2

Software and Computing Steering Committee. Body consisting of representatives from LHC experiments to provide requirements and monitoring to LCG sub-projects.

RTAG

Requirements and Technology Assessment Group. Launching an RTAG on a given domain is the mechanism for defining potential common projects within the LCG project.

SPI

LCG Software Process and Infrastructure project.

2. Project Organization

2.1 External Interfaces

? There are several projects already existing in the LCG Applications Area and SEAL is one of them. Figure 1 shows these projects and their interfaces.

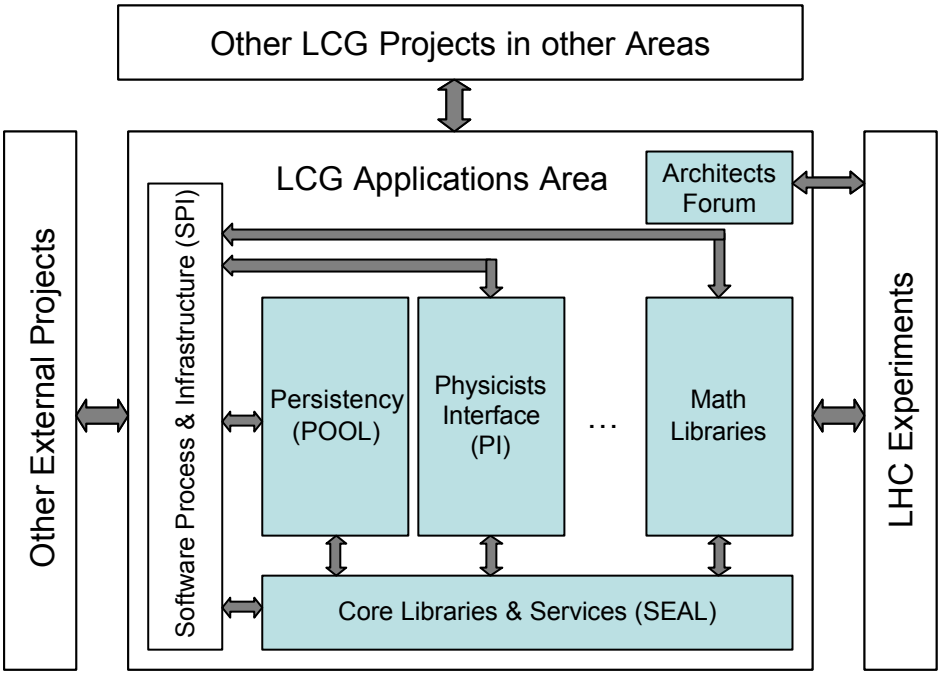

Figure 1 Organisational chart depicting the projects of the LCG Applications Area and their interfaces (arrows)

? Project management. The project reports to the LCG Applications Area manager (T. Wenaus), to the LCG project leader (L. Robertson) and to the SC2 committee (chaired by M. Kasemann).

? Line management. The CERN IT and EP staff of the project report to their corresponding CERN group leader.

? Architects Forum. This forum is the body through which the experiments participate directly in the planning, management, and architectural and technical direction of applications area project activities and in particular this project.

? Customers. In its current definition, the users of the deliverables of the project are all the LCG software projects and any other CERN or non CERN experiment interested in using part or the entire set of core libraries and services.

2.2 Internal Structure

Currently we do not see a need for a complex internal structure of the project. The project will be organized as a number of *work packages* (WP) producing one or more deliverables. Each WP will have a responsible reporting to the SEAL project leader.

During the definition phase of the work packages with their deliverables and responsibilities, the project team will work as a single team without any further structure.

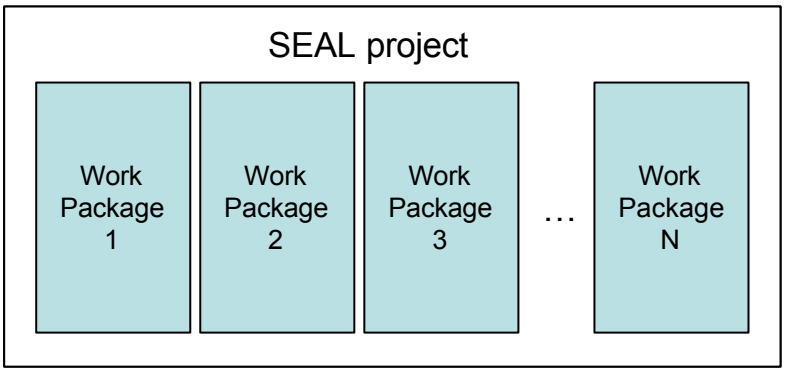

Figure 2 Internal structure of the SEAL project

2.3 Roles and Responsibilities

The roles currently defined in the project are:

? **Project Leader**. The project leader allocates resources, shapes priorities, coordinates interactions with customers and users, and generally keeps the project team focused on the right goal.

? **Work Package Manager**. Responsible for delivering the agreed products of the work package.

? **Developer**. The developer is involved in designing and developing software in all process phases. We do not distinguish designers, implementers, testers, etc.

3. Managerial Process Plans

This section of the Project Plan specifies the project management processes for the project. This section defines the plans for project start-up, risk management, project work, project tracking and project close-out.

3.1 Start-up Plan

3.1.1 Estimates

This plan has been produced by the project leader in collaboration with the initial project team based on the perception of the objectives and the existing experience of similar projects of these characteristics developed in the various core software projects in the LHC experiments.

Weekly status and progress meetings will provide data for improvement of these estimates and for planning the second release.

The schedule will be re-estimated after 6 months or in the event of major changes in the policies regarding the project objectives, such as urgent needs of new LCG projects currently at the RTAG definition phase.

3.1.2 Staffing

? The initial idea concerning staffing is to start with rather small team to define the components that need to be developed and establish the development process and style. The software skills of this initial team should cover various computing domains. In addition, the team members should have strong links to the experiments (our final clients) to ensure that the project deliverables will integrate well with the experiment application software and therefore be easily adopted by them. After this initial phase, the project team will be expanded incorporating new developers and work package managers to develop the final high quality products. The rationale behind this approach is that is it should be easy to agree on a development style and convey the architectural vision to a small number of people rather than to a big geographically distributed team.

? It is desirable that the staff working in the project have a several years of experience in this kind of core software. Ideally, it would be nice if the same people that did this kind of work already in the experiments' core software share their experience and build the SEAL software together.

? The current level of staffing in this initial phase of the project is about 3 FTE. This should ramp to about 8 FTE during the next 6 months just around the time of the release of version v1.

? The expected level of commitment of the staff during the first phase should be more than 50%. This is necessary in order to progress quickly during the initial phase, and to establish the team dynamics.

? The sources of staff
  - Contribution of the LHC experiments to the LCG project (staff funded by CERN and other institutes).
  - LCG funded posts.
? The following table shows the list of current staff of the project and their current commitment to SEAL.

Table 1 Roles and Staffing

Role	People	Commitment	Comments
Project Leader	Pere Mato	50%	also in LHCb
Developer	Jacek Generowicz	70%	also in Geant4
Developer	Massimo Marino	70%	also in ATLAS
Developer	Lorenzo Moneta	50%	also in SPI
Developer	Stefan Roiser	50%	also in LHCb
Developer	Lassi Tuura	50%	also in CMS

3.1.3 Resource Acquisition

The resources to be acquired to fulfill the estimated of 8 FTE will happen in the next 6 months. The acquisition mechanism is mainly based on the goodwill of the people (mainly from the experiments' projects) to join the project if they are interested in developing some of the components.

3.1.4 Project Staff Training

No specific training is required. It is assumed that the interested people joining the project will have sufficient experience in the domain that training should not be necessary.

3.2 Work Plan

3.2.1 Work Breakdown Structure

The initial work packages of the project have been defined and are shown in Table 2 together with a short summary of each one. Later in this section, each work package is described in more detail and the deliverables are listed indicating if it is foreseen for the first release of the software or later releases.

Table 2 SEAL Work Packages

WBS	Name	Summary
1	Foundation and Utility libraries	Set of foundation and utility libraries including an operating system abstraction layer to be used by all other LCG software projects.

2	Component Model and Plug-in Manager	Develop or adapt a basic set of mechanisms and base classes for managing creation of objects (factories), lifetime, multiplicity and scope, component communication and interface discovery. Develop the plugging manager is responsible for loading, activating, deactivating and unloading plug-ins at run-time.
3	LCG Object Dictionary	Develop an object dictionary to provide reflection functionality by complementing the native C++ language features. The two aspects: population of the dictionary and reading the dictionary information through the reflection interface are considered.
4	Basic Framework Services	Develop basic framework services for message reporting, exception handling, component configuration, "event" management, object "white board", etc. Other services candidates to be developed will be identified by the other LCG applications area projects.
5	Scripting Services	Provide the basic infrastructure to support scripting.
6	Grid Services	Provide a common interface to the Grid services to be used by the other LCG application area projects
7	Education and Documentation	Provide coherent documentation for components either developed in the project or adopted from third parties. Prepare tutorials and provide help for integrating project deliverables into experiments applications.

1 Foundation and Utility libraries

Provide class libraries to complement the standard types and utility libraries to be used by all the LCG software projects.

? Main activities and tasks

- Inventory of existing utility classes
- Provide support for *Boost* library. Boost is a utility library (open source project) that is a strong candidate to standardize on. It is intended to become part of Standard Library (STL) in a future C++ standard.
- Participation in the CLHEP project as recommend in the Blueprint RTAG. Prepare proposal for its evolution.
- Develop SEAL *utility* and *system library* complementary to *Boost* and STL using existing code from various libraries in use in the experiments (ClassLib, Gaudi, HepUtilities, etc.).
- Establish guidelines for selecting external libraries

? Proposed v1 deliverables

- SEAL utility candidates inventory
- Support *Boost* library (installation, documentation, etc.)

- Initial version of SEAL system abstraction library
- Initial version of SEAL utility library
- Proposal for external software decision process

? Later deliverables

- Incorporation of the CLHEP evolution decided with the CLHEP editors and LCG architects.

2 Component Model and Plug-in Manager

Develop or adapt a basic set of mechanisms and base classes for managing creation of objects (factories), lifetime, multiplicity and scope, component communication and interface discovery. The plugging manager is responsible for loading, activating, deactivating and unloading plug-ins at run-time.

? Main activities and tasks

- Define component and interface model following the blueprint report guidance. This involves defining interfaces, abstract factories, etc.
- Develop plug-in Manager. Service in charge of managing, querying, [un]loading plug-ins and application bootstrapping (initialization)
- Define "Object management protocol" to define the object lifetime strategy.
- Document Component Model

? Proposed v1 deliverables

- Basic set of interfaces and base classes to support the *Component Model*.
- Initial version of plug-in Manager. This initial version should be sufficient to be used by the POOL project.
- Description of the Component Model and Object Management Protocol

? Later deliverables

- Plug-in Manager with sufficient functionality to be used by experiment frameworks

3 LCG Object Dictionary

Develop an object dictionary to provide reflection functionality by complementing the native C++ language features. The two aspects: population of the dictionary and reading the dictionary information through the reflection interface are considered.

? Main activities and tasks

- Reflection packages (Reflection and ReflectionBuilder), currently part of the POOL project, should be imported and enhanced with some improved functionality with respect to templated types and method stubs.

- Develop tools for populating dictionary from C++ header files, which is required by CMS and ATLAS. Investigate *gcc-xml* technology. This work has originally been carried out in the context of the POOL project.
- Develop gateway of the object dictionary to Python (Python binding). This work is interesting for completeness and as a usability exercise of the reflection interface provided by the dictionary.
- Develop gateway from ROOT to object dictionary. Be able to populate dictionary from CINT (inverse direction to the one developed currently in POOL). This should allow to interact to any ROOT object as if it were defined in the LCG dictionary

? Proposed v1 deliverables

- Reflection packages with small improvements: a) replace static *stub functions* by function objects, b) exploit templates for generation of *stub functions*
- Generation of dictionary from header files (partial C++ support) sufficient for CMS and ATLAS event model
- Python binding to the dictionary using *Boost.Python* technology while waiting for the result of the evaluation carried in the scripting support work package.

? Later deliverables

- Full C++ support for the generation of the dictionary.
- Gateway from ROOT to object dictionary.

4 Basic Framework Services

Develop basic framework services for message reporting, exception handling, component configuration, "event" management, object "white board", etc. Other services candidate to be developed will be identified by the other LCG applications area projects.

? Main activities and tasks

- Develop set of basic services for message reporting, exception handling, component configuration, "event" management, etc. It is understood that more services of common interest will be identified in other projects and they will also be developed.
- Develop object "whiteboard" as described in the Blueprint RTAG report. Study interaction of this component with the persistency services, visualization services and others.

? Proposed v1 deliverables

- Minimal set of basic services sufficient for POOL project: message reporting, exception handling, component configuration.

? Later deliverables

- More complete set of basic services.

5 Scripting Services

Provide the basic infrastructure to support scripting. In particular, bindings for *Python* and *CINT* of the basic services will be needed to provide a "component bus" that allows easy integration of components, possibly implemented in a variety of languages and providing a wide variety of functionality.

? Main activities and tasks

- Evaluate the existing technologies for developing Python bindings (Python extension modules) of C++ classes (SWIG, Boost.Python, SIP...). Define guidelines for developing Python bindings and in particular study the way to solve the Python extension modules inter-dependencies.
- Develop Python bindings following the guidelines for the standard services and utility libraries developed in SEAL
- Enable scripting for application configuration
- Upgrade the existing package for binding ROOT to Python, which enables the interaction with any ROOT class from Python, following the defined guidelines.

? Proposed v1 deliverables

- Evaluation report of the technologies for creating Python bindings. Guidelines for creating Python bindings.
- ROOT python bindings (PyROOT) following guidelines.

? Later deliverables

- Bindings to all SEAL provided services and libraries

6 Grid Services

It is expected that the API to the Grid middleware services will be available as a set of linkable libraries. This work package should provide a common interface to the Grid services to be used by the other LCG application area projects as independent as possible of the middleware implementation.

? Main activities and tasks

- Gather requirements from POOL, PI for GRID-enabled services
- Provide common interface to various Grid middleware

? Proposed v1 deliverables

- None

7 Education and Documentation

Under this work package we group one of the main activities of the project. It is very important for its success that the core libraries and services either developed in the project or adopted from third parties are correctly documented and properly advertised to the other LCG projects' developers and to the

experiments' framework developers which will need to integrate the experiment specific code with the provided LCG components.

? Tasks
  - Write the documentation of the software components developed by project.
  - Facilitate access to the documentation of third-party software components and present it in a coherent way with the rest of the documentation.
  - Develop tutorials presenting the project deliverables.
  - Help incorporating SEAL components into LCG projects and experiment frameworks.

? Proposed v1 deliverables
  - Documentation of all delivered components in version v1

3.2.2 Schedule Allocation

? **March 2003 – v1 alpha version**.
This version is defined as including all the essential components with their functionality sufficient for the needs of the other existing LCG projects (e.g. POOL).

? **June 2003 – v1 beta version**.
This version is defined as including all the essential components with their functionality sufficient to be adopted by experiments frameworks if they wish.

? **December 2003 – v2 version.**
This version is defined as the first complete version including implementations for all components known at the time of this plan.

3.2.3 Resource Allocation

Table 3 shows the estimates of the required FTE for each of the defined work packages. Concerning the currently available resources, the total number is correct but their sharing between the work packages is just a guess since we have not yet assigned people to work packages at this phase of the project.

Table 3 Resource allocation

WBS	Name	FTE (available/required)
1	Foundation and Utility libraries	0.5 / 1.0
2	Component Model and Plug-in Manager	0.5 / 0.5
3	LCG Object Dictionary	0.5 / 2.0
4	Basic Framework Services	0.5 / 1.0

5	Scripting Services	0.5 / 1.0
6	Grid Services	0.0 / 1.5
7	Education and Documentation	0.5 / 1.0

3.3 Project Tracking Plan

3.3.1 Requirements Management

- After each public release of the software every 3 months, the requirements can be reviewed by the experiments. If changes are needed, the proposed modifications will be added into the *bug tracking system*[1].
- The impact of the proposed requirements changes will be assessed based on the product scope and quality, and on project schedule, budget, resources and risk factors.
- Using the *bug tracking system* we expect to be able to trace the changes in the requirements to the versions of the software implementing such changes.

3.3.2 Schedule Control

- The project has defined a number of major milestones.
- The progress of the work completed will be measured at the major and minor milestones.
- The project plan will be controlled and tracked quarterly.

3.3.3 Budget Control

- No special need to control the budget.

3.3.4 Quality Control

- To measure and control the quality of the work and the resulting work products will be done by making statistics of the number of defects of the software (bugs) using the *bug tracking system.*
- The quality control processes defined in collaboration with the SPI project will be applied to the project.

[1] GNU Savannah service provided by SPI

3.3.5 Reporting

? The progress of the project will be controlled and tracked quarterly. Periodic status reports will be presented in the Application Area meetings and the PEB meetings when required.

? The project plan should be revised every six months in order to reflect the changes in requirements and priorities, and the evolution in the level of staffing. The updates will be presented to the SC2 together with the tracking of the existing plan.

3.3.6 Project Metrics

? We still need to define the metrics which would provide meaningful measures of the quality, usability, maintainability, modularity, etc. of the foundations libraries and basic framework services.

? The frequency for collecting metrics data is defined to be at the same frequency as the public releases. Every three months.

3.4 Risk Management Plan

We have not defined a risk management plan yet. This plan should identify, analyze, and prioritize the project risk factors. We need describe the impact of the following risk factors:

- risks in the customer-project relationship,
- technological risks,
- risks caused by the size and complexity of the product,
- risks in the development and target environments,
- risks in personnel acquisition, skill levels and retention
- risks to schedule and budget, and
- risks in achieving customer acceptance of the deliverables.

3.5 Project Closeout Plan

We have not defined a project closeout plan yet.

4. Technical Process Plans

4.1 Process Model

The software development process to be adopted by the project should follow the guidelines described in *Thoughts on Software Process [2]* by T. Wenaus. These guidelines are a mixture of the useful elements meeting the needs of the LCG software taken from two well known and established software processes: Extreme Programming[2] and Rational Unified Process[3].

Summary of the good practices:

? Planning
- Release planning creates the schedule
- Make frequent releases with small functionality increments
- Divide the project into iterations

? Designing
- Design a Component Architecture
- Design as simple as possible, but no simpler
- No functionality is added early
- Refactor whenever and wherever possible

? Coding
- The customer is always available
- Code must be written to agreed standards
- Integrate often
- Use collective code ownership
- Leave optimization until last

? Testing
- All code must have unit tests
- All code must pass all unit tests before it can be released
- When a bug is found, tests are created
- Acceptance tests are run often and the score is published

4.1.1 Iterative Development

The project will work in tight, iterative development cycles: release early and release often. The diagram in Figure 3 tries to be explicit about what the planning, development and feedback cycle could look like.

[2] http://www.extremeprogramming.org/

[3] http://www.rational.com/products/rup

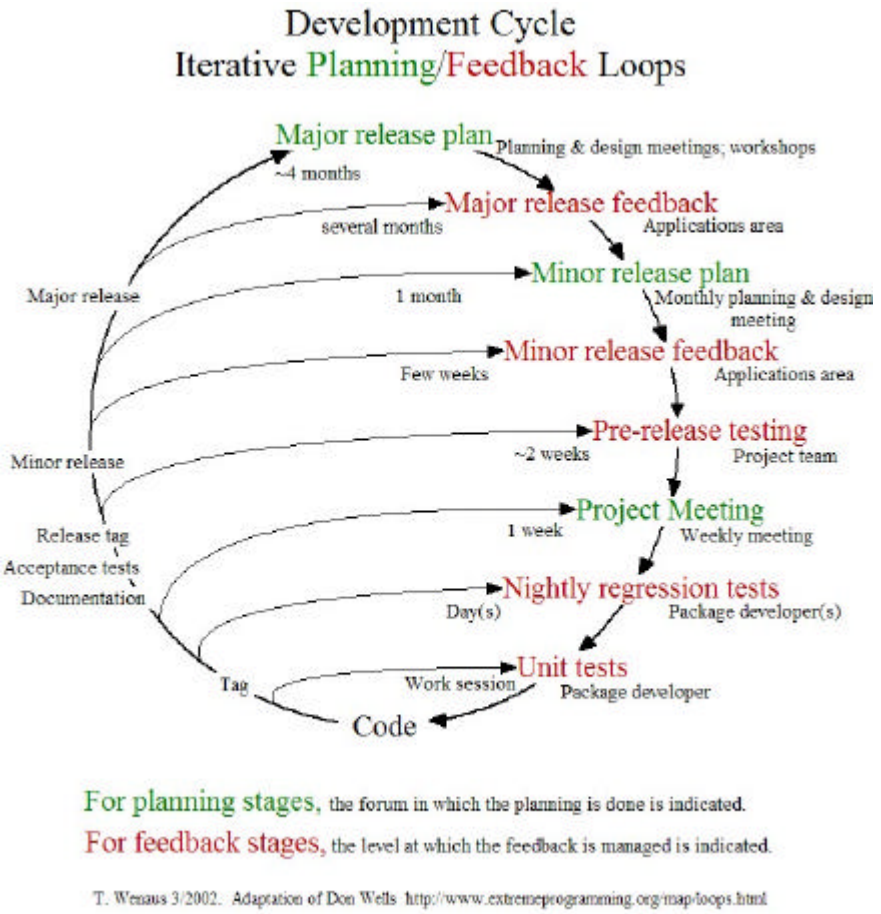

Figure 3 Development Cycle proposed for LCG Application Area software projects

4.2 Methods, Tools, and Techniques

Table 4 contains the initial list of methods and tools going to be used for the development of this software project. These tools are provided or going in the near future to be provided by the Software Process and Infrastructure project (SPI). In addition to the tools themselves, SPI is providing a number of guidelines and templates to support the software process: coding guidelines, repository structure guidelines, configuration management guidelines, testing guidelines, documentation templates, web page templates, project plan templates, etc.

4.3 Infrastructure

The infrastructure needed by the project in terms of hardware (code repository servers, build servers, test servers, web servers, software distribution area, etc.), operating system, network and software licences are going to be provided by SPI project.

The facilities required to conduct the project, which includes desktop workstations, local area networks, desks, office space and others, are going to be provided by the *Line Management* and CERN.

Table 4 List of Tools going to be used during development

Design tools	Together
Code Management	CVS
Conf. Management	SCRAM
C++ Compilers (ISO standard)	Windows: Visual C++ Linux: gcc
Code Editors	Linux: emacs
Debuggers	Windows: DevStudio Linux: gvd (gdb)
Testing	CppUnit or Boost.Test
Memory Leaks	Linux: valgrind
Mark-up Language	XML, Xerces-C
Scripting Language	Python
Documentation	Manuals: Word, DocBook Source Code: Doxygen, lxr
Bug Tracking	Savannah

4.4 Product Acceptance

The goal of the project is to provide foundation software for use by other projects within the LCG. The requirements will therefore continue to evolve, and the concept of final product acceptance is therefore not meaningful.

For intermediate releases, specific feature requests made by other projects should be accompanied by a set of acceptance tests, which define the criteria by which the SEAL deliverable shall be judged to have been completed to satisfaction.

5. Supporting Process Plans

5.1 Configuration Management

Configuration Management deals with defining configurations, building and labelling, and collecting versioned products into consistent sets and maintaining traceability between these versions. We will use CVS and SCRAM tools to fulfil all configuration management needs of the project. Currently, we do not have a *Configuration Management Plan* in place. In the following we describe some elements of the configuration management plan such as the product structure and identification of the constituent configuration items and some procedures.

? The product structure is depicted in Figure 4. The central configuration entity is a *Package*, which is the minimal entity that makes sense to release and associate a version (CVS tag). All the packages (delivery products) of the SEAL project are organized in a two level structure.

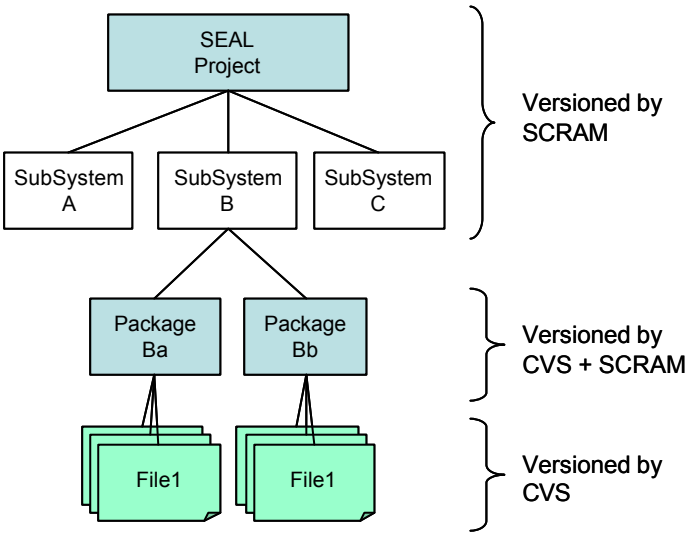

Figure 4 Product structure of the project with respect configuration management

? Package versions will be identified with a string of the format "PackageName_Major_Minor_Patch". Where "Major", "Minor", "Patch" are version numbers with a specific semantics. The "Major" number is going to change for major releases of the software, which may require a major re-coding of customer software because the interface or the use model is changed. The "Minor" version number indicates new or a change in the functionality but the existing customer software is compatible at source level with the previous version. The last version number, "Patch", indicates bug fixes, no new functionality being added.

? The release management will be done in various steps:

- The developer of a package can apply an *internal* tag as a declaration on the part of the developer that the package is tested and working.

- The package coordinator has the right to apply a *release* tag indicating that the package is tested and ready for release integration.
- The release manager will collect all the *release* tags for all the packages constituting the project, and will integrate them to produce a project release.

? Change requests will be handled using the *bug tracking tool*. Requests will always refer to a given version of the software. The changes in the software will be explained and identified by the change request identified in the *ChangeLog* file. This file exists in each package and therefore is tagged with the rest of the files constituting the package.

5.2 Verification and Validation

Tests will be written together with the code. Unit tests should validate expected functionality at the level of individual classes or small groups of collaborating classes. Regression test cases at levels higher than unit tests, tests involving package and component interactions and system level tests, should be written based on problems found in the pre-release QA process user reported problems. System level tests should be based on requirements' test cases that have failed in the past. These should be collected cumulatively to assemble a comprehensive regression test suite. Regression testing involves applying a standard test suite to a candidate release to find bugs introduced since the last release (i.e. to spot regression), including bugs seemingly unrelated to the changes introduced. They test changes in integration behavior: they are typically large-grained, and test for unexpected consequences of integrating software components.

In addition to validation based on the different types of tests described, we plan to perform:

- Validation of coding standards compliance
- Dependency analysis
- Memory leak checking

All levels of testing (unit tests, component level tests, component interaction tests, system level tests) should be run as part of automated nightly (and pre-release) builds.

5.3 Documentation

Standard pieces of documentation for the project are:

- Installation guide
- User manual with code examples (tested as part of release acceptance)
- Design documentation
- Auto-generated web-based documentation and code browsing
- Release notes
- Chapters in the LCG Workbook

5.4 Quality Assurance

We have not defined a specific Quality Assurance plan. QA elements are covered in various sections of this document. Here is quick summary:

- Documentation. Documents foreseen for the project deliverables: design, installation guide, user manual, generated reference from code and release notes.
- Standards and Guidelines. Set of guidelines and templates are being provided by SPI. This includes the coding guidelines and the development workbook.
- Metrics. Metrics are going to be developed by SPI. Some examples are: number of implemented tests, test results, validation results, nightly build performance (failure rate), bug counts in released code (from bug reports), time to bug resolution, milestone performance, amount of invested effort, coding standards compliance, compile performance (warnings, etc.), number of dependencies, kLOC, commit count (frequency), date of last commit, developer count (cvs committers)
- Review and Audit Plan. Quarterly project status reports. Design, requirements and code reviews every major release.
- Evaluation and Tests. Writing tests is part of the software coding. Various kinds of tests are foreseen: unit tests, regression tests, system level tests. In addition we plan to check for memory leaks, package dependencies and conformance with coding rules.
- Problem Resolution and Corrective Action. The bug tracking tool will be used to report and track problems up to their resolution.
- Tools, Techniques and Methods. A number of widely used tools will be used all the way through the development process. They are listed in section 4.2.
- Configuration Management. The combination of CVS and SCRAM will provide us with the necessary tools to support the configuration management activity.
- Training. Training is part of the deliverables of the project in form of documentation and tutorials.
- Risk Management. No specific risk management plan defined.

5.5 Reviews and Audits

Overall project reviews and audits are not currently planned. The quarterly reports to SC2 would be an opportunity for requesting a review of the project.

After each major release would be an opportunity to perform joint customer-project reviews (to confirm that the customers' expectations are met), design reviews by the software architects (to verify that it complies with the

architectural vision depicted in the Blueprint RTAG) and code walk-throughs (to assess the code quality and compliance with the coding rules and guidelines).

5.6 Problem Resolution

Problems will be reported and tracked using the *bug tracking tool* provided by SPI project. Nothing special is foreseen in addition to the standard process of analyzing, prioritizing and solving the problems reported and trying to give full satisfaction to the customers.

5.7 Process Improvement

Any opportunity to improve the process will be taken into account. The project status reports to the reporting bodies and to the customers, the reviews with customers and the reviews with the software architects and developers will certainly offer these opportunities to identify areas where some changes in the software process will be beneficial. These changes can be immediately applied if there is no serious disruption of the ongoing project

In addition to this general statement, we have not defined a specific process improvement plan.