A Process for ATLAS Software Development

Authors : Atlas Quality Control Group
M. Asai, D. Barberis (chairman), M. Bosman, R. Jones, J.-F. Laporte, M. Stavrianakou

Abstract

This document proposes an outline of the ATLAS software development process. It defines the scope of the software concerned and the software quality criteria which are considered the most relevant to ATLAS.

Version : 2.0
Date : 26 September 2000
Status : Final

1 Introduction

The purpose of this document is to define the process to be used for the development of ATLAS software and the global requirements to the Quality Control (QC) process. The organisation and mechanisms needed to implement the software process are outlined. This document is addressed to all members of the ATLAS community with the intention of reaching a broad consensus on the definition and implementation of the software development process.

Software quality has to be assured in order to achieve the goal of ATLAS to produce correct physics using the resources efficiently; software quality is equally important in order to maintain the software throughout the lifetime of the experiment. All software projects of such size and complexity, in high energy physics (HEP), other research fields and industry, have acknowledged the importance and benefits of implementing a development and quality control process from an early stage. It is not only beneficial for the quality of the software but also for improving the skills of the developers.

This document is produced by the QC Group that has been created as part of the ATLAS Computing Action Plan [1] put forward by the ATLAS management following the computing review of ATLAS [2]. The QC Group defines the software development process and the procedure to ensure its application. This procedure will replace the ATLAS Software Process (ASP) [3].

Ensuring the quality of the software was one of the main goals of the ASP. The aim of the QC Group is to ensure that this goal will be fully endorsed by the community in the current and future software activities. In addition, quality assurance should be an assistance to developers rather than a constraint or additional burden. The procedure outlined in this document will evolve as the ATLAS software community gains experience with the software quality model.

The ATLAS experiment is a very large community. Many institutes, widely distributed geographically, contribute to the software. Usually, there are only a few developers in each group. Good communication channels and availability of information have to be assured. In addition, the software is going through a transition phase to OO/C++. Many contributors are newcomers to C++, to OO and also to QC procedures, and will go through a learning phase. The C++ language and Object-Oriented approach introduces new concepts that require more guidance for software developers and the elements of the QC process, such as the Coding Conventions, are intended to be an aid. The chosen strategy for QC must take these facts into account and the policy should distinguish the long term target from the transition phase.

The software considered in this quality model is the offline software, i.e. all the software necessary for data processing and analysis offline. It also includes event filtering and physics monitoring on-line as well as the simulation of the LVL1 and LVL2 triggers. The model can be adapted to be applied to external software.

The document is organised as follows. Section 2 gives an overview of the ATLAS software development organisation and a brief description of the ATLAS environment. Section 3 outlines the model proposed for the software quality assurance. Section 4 lists some recommendations on the software development process to be adopted by ATLAS and the associated tools. Section 5 lists the main criteria that will be used to evaluate the quality of the software. Section 6 outlines the implementation of the software quality model.

2 ATLAS Software Development Organisation

The current organisation of the ATLAS offline software development working groups foresees basically four levels:

- Core software. This consists of the global framework plus a few packages for geometry and event data access, graphics and interfaces to external software.
- Detector software. It comprises all the detector simulation software, the treatment of raw data (decoding, alignments and calibrations) and the first stage of event reconstruction (tracks in the Inner and Muon Detectors, clusters in the Calorimeters).
- Combined reconstruction software. It includes particle identification, refined calibrations, jet tagging, vertex assignments. Its output is an event containing vertices, jets and particles.
- Physics analysis software. This is the last step of event selection and processing, which will be repeated many times with very different analysis programs.

Each piece of software, from a small inline function to a large pattern recognition package, is usually developed by a very small number of authors — most of the times just one. This development nevertheless does not (and should not) take place in isolation, but rather in the framework of the activities of a software development working group. It is the working group that coordinates the development activities and, in the end, holds the responsibility for each piece of software it has developed. In a sense, the working group "owns" a piece of software.

It is therefore rather obvious to assign to the working group that owns a particular piece of software also the responsibility for its development process and quality control. The working group must discuss the requirements, design, implementation and documentation of each piece of software under its responsibility. It must organise the appropriate reviews at each step in coordination with the Quality Control group.

3 Software Quality Model

The long-term target of the software quality policy is to verify that the software is well adapted to its purpose by examining its design and structure through organised inspections and reviews and to ensure the code is correct, robust and performant via testing procedures. The availability of good documentation is considered essential.

A model has been developed in order to reach the long term goal, allowing at the same time the flexibility needed for the evolution of the software during the transition period. In the software "onion" model, "kernel" software is in the centre and outer layers are made of domain specific, detector specific then individual physicist's software. In this model, the more widely used and long-living software is closer to the core. This software, due to its importance to the experiment as a whole, should satisfy stricter requirements than an individual's private software. Software for personal usage can be freer but the authors should be well aware that if they want to promote their software to a higher status (for example from exploratory analysis to a tool of a working group), then they will have to comply with stricter conditions, especially in terms of documentation and coding guidelines. This becomes imperative when the software is used for analysis that is intended for the publication of physics results.

There are two aspects in the quality of the software that should be distinguished. The first addresses the technical aspects of the code quality, such as speed of execution, use of memory, portability etc. The second aspect concerns the physics performance of the algorithms such as efficiency, resolution of computed physics quantities, tails in distributions etc. Both aspects are addressed in this document.

4 Software Process

4.1 Modelling Language and Development tool

Recently, UML [4] has become the standard language for object-oriented analysis and design. We consider its use to be mandatory in all Object Oriented Software documentation. A CASE tool, based on UML, like Together [5] can be very useful for OO software development.[1] Given the rapidly-changing software development environment, we recommend that the performance of such tools be continuously monitored by the Computing Technical Group and that an ATLAS configured tool be made available to the collaboration.

4.2 Methodology

Different methodologies providing useful guidelines for the software development exist. Among these, the Unified Software Development Process (USDP) [6] seems to be a pragmatic approach, well convenient for HEP software. It defines a incremental iterative process allowing implementation and testing at a very early stage of the development. This process is decoupled from the issue of documentation and code acceptance procedure. We therefore recommend USDP as a general guideline.

The USDP/UML combination, together with a development tool like Together, can provide a well-integrated software development environment; this is needed in order to be able to develop coherently software in a geographically distributed collaboration.

5 Software Quality Criteria

A list of criteria that define the quality of the software is given below.

5.1 Quality of design

5.1.1 Definition

The code should have a clear design, be modular and be compatible with the global ATLAS software architecture.

5.1.2 Explanation

The aspects that will be considered are not only the internal design of the package but also its integration in the overall ATLAS software environment. Strict criteria will be applied to the quality of the interfaces already during the transition period.

1. We are aware that most CASE tools are available only under licence; we therefore recommend that such licences be negotiated by the CERN/IT management on behalf of all participating institutes.

5.2 Documentation

5.2.1 Definition

Each piece of code has to be accompanied by the appropriate documentation. The documentation should consist of:

- a document stating clearly what is the task the code should perform, and which is the method (algorithm) used to achieve this task;
- a software design document;
- a description of input and output interfaces to the piece of code (“users' guide”);
- an example of usage and all accessory files to run the example.

The documentation must be kept up-to-date with the software. Documentation for a given release should be available from the CVS repository and follow automatically the updates of the corresponding software.

5.2.2 Explanation

The task to be accomplished by the piece of software and the algorithm(s) used should be clearly described in full detail.

The design should be documented both in English and (for new software) also in Unified Modelling Language (UML) [4] form: all classes and their interrelations should be described. The plain language documentation should contain at least the high-level concepts of the package design. The UML documentation should have a detailed description of the implementation, possibly kept up-to-date using automatic tools. Existing C++ software should be “reverse engineered” to document the design as far as possible.

Special emphasis should be given to the “users' guide” that should contain a detailed description of all input and output parameters, as well as any other information directly useful for the integration of the given piece of software into the existing environment.

A code testing plan should be provided. It should cover the usual pitfalls of C++ code, memory leaks, timing measurements etc.

The example of usage should contain a reference set of results for given input conditions. This should allow the user to check that the package is being used correctly.

After the transition period, complete documentation will be required for all publicly available pieces of software.

5.3 Coding conventions

5.3.1 Definition

Coding conventions are meant to help C++ programmers to meet the following requirements on a program:

- be free of common types of errors;
- be maintainable by different programmers;
- be portable to other operating systems;
- be easy to read and understand;
- have a consistent style.

5.3.2 Explanation

The coding conventions are adapted from the SPIDER project [7]. Code examples or further clarification and justification are given wherever necessary. An automatic tool to check at least the most important coding conventions will be provided.

5.4 Robustness

5.4.1 Definition

The robustness encompasses all the “technical” qualities that should characterise a piece of the code. It should use the computing resources efficiently, be stable, and run in the various required environments.

5.4.2 Explanation

The code should be correct and efficient in terms of memory usage and execution. Its execution should not adversely affect other software modules.

Some simple metrics, like code size and code/comments ratio, will be applied to the code. More complex OO metrics will be considered as the complexity of the software increases.

The code should compile and run correctly on all the official platforms. The code should not crash but rather give warnings even in case of incorrect running conditions.

5.5 Maintainability

5.5.1 Definition

The maintainability of the code is related to the effort needed to identify, make and validate necessary modifications.

5.5.2 Explanation

The diagnosis of problems and the identification of the part of code to be modified should be achievable with reasonable effort by a person other than the author. To this end, the code should be readable and internal diagnostic (e.g. tracing facilities) must be provided when possible. Changes of environment (e.g. running on another computer platform or in a different context such as offline reconstruction or the event filter) should be possible using the means provided for this purpose by the software. Adapting the testing procedure to the new environment should be reasonably straightforward.

5.6 Performance

5.6.1 Definition

The algorithms(s) provided by a piece of code should satisfy the required physics performance criteria.

5.6.2 Explanation

The physics quantities computed in an algorithm should comply with the standards of precision and efficiency required in the context in which it is used. Other factors play a role here, such as the trade-off between efficiency and speed or effect on the precision of the quality of calibration or alignment constants available at the time the calculations are performed. The subsystem and combined performance groups have to define the requirements on the performances and organise the evaluation procedure.

6 Implementation of the Software Quality Model

There are two elements in the implementation of the software quality model: provide support to the developer to check and improve his/her own software; set up a mechanism of validation of software packages by a process of peer reviews.

The support to the developer includes automatic checking tools for coding conventions, skeletons of code for the most common tasks and templates for all the documents that are required.

The validation mechanism will be achieved through a combination of inspections, reviews and tests organised by the working groups that “own” a particular package. The working group must organise for each package the appropriate validation procedure:

- appoint a group of reviewers;
- make sure that the reviewers receive the documentation ahead of the review/inspection;
- organise the actual review/inspection;
- feed back the results to the whole community.

It is advisable that one member of each working group take charge of this organisation, initially with the help and advice of the members of the QC group. Downstream users of the package under review should be also involved in the validation process. In order not to introduce too much overhead, the reviewing process should normally not take longer than the time between software (or system) weeks.

A testing plan describing the features to be tested and the results of the tests will be part of the validation process. The integration of the package into the existing software framework should be thoroughly tested.

All code must be shared through the ATLAS software repository, following the ATLAS offline repository policy[2]. Only the package versions that have passed through the QC valida-

2. See http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/HELP/cvs_srt/repository_policy.html.

tion procedure can be included in the official software releases[3]. The strictness of the validation criteria will have to evolve during the transition period.

7 Summary

Software quality has to be assured to reach the goal of ATLAS to produce correct data using resources efficiently and be able to maintain it throughout the lifetime of the experiment. The general policy for the ATLAS software quality control has been defined and the global requirements on the quality control process have been set in this document. In addition to quality assurance, the aim of the process is to provide a useful aid to the developer.

The software considered in this quality model is the offline software, i.e. all the software necessary for data processing and analysis offline, including event filtering and physics monitoring on-line. It can be adapted to external software.

Both coding quality and physics performance aspects are considered. The list of criteria considered are: clarity and structure of design, documentation, compliance with coding conventions, robustness, maintainability and physics performance. The implementation of the software quality model has been outlined.

8 References

[1] P. Jenni and T. Åkesson, ATLAS Computing action plan,

http://www.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/action_plan

[2] H. A. Neal *et al.*, Report of the ATLAS Computing Review Committee (AT-RO-MR-0001).

[3] S. M. Fisher *et al.*, The ATLAS Software process,

http://www.cern.ch/Atlas/GROUPS/SOFTWARE/OO/asp/ref/ref.html

[4] G. Booch J. Rumbaugh and I. Jacobson, The Unified Modelling Language User Guide, Addison-Wesley, 1999.

[5] http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/OO/tools/case/Together/use.html

[6] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

[7] SPIDER Project: C++ Coding Standard (SPIDER-SDP-PR-0003-TRP)

http://spider.cern.ch/Processes/Programming/CodingStandard/c++standard.pdf

3. See also the ATLAS offline release policy:
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/HELP/librarian/ReleasePolicy.html.