SOFTWARE DEVELOPMENT AND TESTING: APPROACH AND CHALLENGES IN A DISTRIBUTED HEP COLLABORATION

D. Burckhart-Chromek, CERN, Geneva, Switzerland

ATL-DAQ-CONF-2008-008
06 July 2008

Abstract

In developing the ATLAS [1] Trigger and Data Acquisition (TDAQ) software, the team is applying the iterative waterfall model, evolutionary process management, formal software inspection, and lightweight review techniques. The long preparation phase, with a geographically widespread development team required that the standard techniques be adapted to this HEP environment. The testing process is receiving special attention. Unit tests and check targets in nightly project builds form the basis for the subsequent software project release testing. The integrated software is then being run on computing farms that give further opportunites for gaining experience, fault finding, and acquiring ideas for improvement. Dedicated tests on a farm of up to 1000 nodes address the large-scale aspect of the project. Integration test activities on the experimental site include the special purpose-built event readout hardware. Deployment in detector commissioning starts the countdown towards running the final ATLAS experiment. These activities aim at both understanding and completing the complex system, and help in forming a team whose members have a variety of expertise, working cultures, and professional backgrounds.

INTRODUCTION

ATLAS is one of the LHC experiments at CERN that will start taking data in 2008. The ATLAS Trigger and Data Acquisition (TDAQ) system [2] will consist of more than 2000 PC nodes, which take part in the control, physics data readout, event building and event selection operations.

Following a Software Development Process (SDP) is well established in the software industry. Various techniques are being employed depending on factors like project size, project life time, and team size. Traditional models derived from the Waterfall Model and the Spiral Model are being applied for mission critical long term projects. Agile methods are being used for projects with a shorter life time and rapidly changing requirements like Internet oriented tasks.

The TDAQ development for the control and physics data readout software spreads over a time span of a decade with a large number of collaborating institutes from all over the world. The project has applied an SDP that is based on the traditional methods but needed significant adaptation while retaining emphasis on result-oriented aspects such as requirements specification and testing. Sub-systems working in small groups apply ideas taken from agile methods at times.

SDP OVERVIEW

The Traditional SDP

The traditional SDP methods impose a structure on the software product development. A list of distinct development phases provides the frame of a model. The software development models describe a variety of tasks and activities that take place during the process.

Most commonly *Domain Analysis* is followed by a *Software Element Analysis* or *Brainstorming* phase before *Specifications* are gathered during the *Requirements* phase. Once the requirements have been reviewed, *Software Architecture* and *Design Phases* serve to describe the project or task in an abstract design. *Implementation and Testing* including *Review* and *Inspection* will bring the product to life. *Documentation* and *Software Training* of the product introduce the newcomer to the product. *These phases are followed by the Deployment, Support and Maintenance,* which continue throughout the lifetime of a software component to adapt it to new operating system releases and compilers. This Waterfall Model, published first in 1970 by W. W. Royce can be enriched by feedback paths to one or more previous phases suggesting iterative development. These aim at improving the product and responding to the evolution of the software system in which the component is embedded. The Spiral Model defined by Barry Boehm in 1988 introduces risk management. Each iteration phase is carefully planned from the start to increase the system's functionality and complexity.

Agile Methods

Agile software development is a conceptual framework that embraces and promotes rapid evolutionary change throughout the entire life-cycle of the project. In Extreme Programming (XP) [3] for example the development iterations take a few weeks after which the new software is released. Document writing is largely omitted, face-to-face communication is preferred and pair programming is favoured. Interestingly, testing is strongly emphasized. The test program is written before the code of the product is produced. XP is aimed at small projects where programmers are located very closely together, in most cases in the same room. It does not contain a concept which foresees collaboration with remote sites. A big project can be broken into distinct working areas and can make use of some of the advantages of an agile method like XP if the boundaries of a sub-project and its required functionality are well-defined and if it involves few developers.

SDP CHALLENGES FOR ATLAS ONLINE IN THE HEP ENVIRONEMNT

ATLAS will have a lifetime of 15 years and the development of the TDAQ project has started 10 years before the start-up of the experiment. The development team comprises physicists and software engineers with varying professional backgrounds who often have short term contracts or additional duties at their home institutes.

Over 60 institutes are participating in TDAQ and thus the 400 team members are located worldwide. The collaborators accept that they may have to travel to the CERN site, but the use of collaborative tools is indispensable. Unlike industry, there is no strong executive power. Consensus has to be reached by agreement amongst the participants. Traditionally, physicists are reluctant if faced with a working framework, rules to obey and suspicious about being controlled in their work. They are creative but have little training in modern programming language. It was necessary to adapt the traditional SDP, lighten it and introduce it gently to this working environment which lacks homogeneous starting conditions. The project benefited from a supportive framework and the integration activities which provide the SDP methods at given check points.

Introducing the SDP

The ATLAS online activities are grouped into the infrastructure software which includes control, configuration and monitoring services, the readout and transport of physics data, the event selection and the event building tasks. An initial infrastructure system with reduced functionality and scope was needed early in the preparation phase of the project for detector test activities in labs and at the test beam site. Twelve packages with 30 components were developed by ten team members working only part time on the project.

By applying a SDP the development has been divided into sequential phases intended to help pace and organise the work. Typically, a single institute has taken responsibility for developing a component and thereby simplifying communication and reducing travel. Each development phase has been defined to produce an obvious deliverable, i.e. document and/or code. Each deliverable from each phase was reviewed before progressing to the next phase. The phases, after an initial brainstorming phase, are to: collect requirements; identify and evaluate candidate technologies and techniques capable of addressing the common issues identified from the requirements; produce a design for each component covering the most important aspects; refine the design to add more detail; implement and unit test according to the design; integrate with other components. Phases were not taken up strictly sequentially but overlapped occasionally. Design could start when the prime requirements were known and code was written for evaluating candidate technologies. The use of a prototype helped clarifying the requirements.

Informal reviews took the form of presentations followed by discussions during open meetings with all developers involved in the project. The review of each deliverable in the project at each of its phases has lead to a coherent set of end-product components. Modifications of a component due to the evolution in ideas or due to technical constraints could be accommodated in agreement with other components and their developers. A drawback was that the reviews were not performed thoroughly and team members often did not find the time to write or read the documents before the meeting. Formal Inspection was introduced to get reviewers to pinpoint their work and to help them justify spending valuable working time on such tasks.

Widening the Scope

When the integration of the various TDAQ sub-groups started, a forum was build with one representative per sub-group. The SDP was documented in web pages through a team effort. Guidelines were summarized in short and easily digestible checklists and document templates and example documents were provided. Training for software inspection took place. Newcomers were integrated into the software development activities for example as peer reviewers. The members of the forum acted as link persons to their respective sub-group.

The integration of the sub-system functionalities to encompass the software of TDAQ and thus the integration of the development teams were exercised in steps. The SDP which had been applied in the infrastructure group was followed in parts, notably for components which interfaced between sub-systems. Formal inspection with a review team composed of members from several sub-systems brought the breakthrough and acceptance in the community for technically important and politically prominent components like the control software and the physics selection software.

With time, the principle of the development process and the guidelines had been assimilated. The project had moved to the implementation, testing and deployment phases. The regular integration testing activities gave the opportunity to continue integrating newcomers who had gained experience in a different setting.

SOFTWARE INSPECTION

Principle Aims

Inspection is performed before testing as part of the **Defect Detection** Process and it complements testing. Documents are checked for cleanness and consistency against rules. The objective is to identify and correct major defects in the candidate product before releasing it from the current development phase.

The **Defect Prevention** Process is concerned with learning from the defects found, and suggesting ways of improving the processes to prevent them from recurring in the future. It involves process analysis, which is carried out off-line from the normal inspection of specific documents. Team participants benefit from the experience gained during the inspection and subsequently improve their own work while the inspection process is improved from participant's suggestions and according to changes in technology.

On the job training is a valuable benefit of a dynamic and open inspection process. It provides implicit integration and education of people who are new to the project. Process guidelines and checklists are available for convenient entry into the project, while being open to easy modifications and additions of new ideas.

The Inspection Process

The inspection process [4] is managed by an inspection leader who chooses the reviewers, prepares the logistics and runs the kick-off and the logging meetings. The inspection process is based on the method from Tom Gilb and Dorothy Graham [5]. While formal software inspection relies on a given framework, it was a guiding principle that everything be allowed which may help to improve the product, the overall production process, communication amongst project participants and the inspection process itself; at the same time keeping consistency and improving efficiency. Formal software inspection had served as a major vehicle to build the rules, guidelines and working habits, to familiarize team members with them and to experience the benefits of helpful criticism and improvement suggestions by colleagues. Logging meetings helped to integrate remote team members and clarify deliverables.

Light Inspection

Once inspection had been accepted in the development community, a lighter form of inspection which would take peer reviewers less time and reduce the work load to organize the inspections was sought. Logging meetings were replaced by the use of electronic communication tools. This helped saving time while retaining the primary benefits of inspection. Documentation and checking followed the same in house standards, rules and checklists as for the formal inspection. For example, once having become familiar with the specific style of writing requirements, it could be written quickly if only the technical specification were clear. It was easy for colleagues to read the document and understand subtleties in the expressions. This form of light review was applied when team members who were familiar with the process were involved. It was not beneficial in cases which included neighbouring sub-systems and review members who were not familiar with the inspection process because there was too much room for misunderstandings. Preference for formal inspection was also given when agreement on a prominent component by several sub-systems or users was sought.

Experience

Requirements inspection was found to be the most important. Requirement inspection is done because according to experience in industry it takes about one hour to find a major defect by inspection at an early stage and about nine hours when testing. Requirement inspection is also the least time consuming because no or few mother documents must be read and they are generally only a few pages long. They turned out to be useful to re-visit and clarify strategy and goals.

Design inspection is the hardest to perform. It was found to be difficult to define a good set of guidelines which is not trivial but also not too restrictive.

Code inspection is the most time consuming one. Many documents are involved: code must be checked for internal consistency and against coding rules; the users' guide and the implementation documentation must be inspected and compared against the design and requirements documents. Automatic checking tools [6] were employed. Sampling was performed in most cases and is recommended to be undertaken regularly by concentrating on critical areas.

TESTING AND DEPLOYMENT

Functionality and verification testing is performed throughout the software lifecycle. New components are tested according to a test plan, preferably by non-authors. Software release testing is performed involving the overall system before integration tests are conducted in test labs at remote institutes and at CERN.

Test-ware is written in small units and is run separately grouped. The tests standardize on command, output and exit codes. They are part of the software repository and follow the evolution of the component. Emphasis is put on critical areas and boundaries. Testing tools are used for code coverage and memory leak checking.

Large Scale Performance Tests

Large scale performance tests [7] have been conducted at the CERN LXBATCH farm. Starting in 2001 by exercising the TDAQ infrastructure on hundred nodes, the scale grew to the use of 1000 dual CPU nodes including most of the sub-systems in November 2006.

On large scale, trend analysis of performance data allows identifying critical areas. Rare problems occur more often and become reproducible. Process communication between several thousands of processes is exercised. The scalability of the state machine for the control of TDAQ is verified. Multiple simultaneous database access is tested. Variants introducing intermediate server levels are studied.

Experience showed that the complexity of the system and the potential for problems grow exponentially with scale. Further development concentrates on improving fault tolerance and stability of the TDAQ system and of the farm management.

The Pre-Series Setup

A dedicated test system of 80 PC's, the so called "pre-series" setup [8] together with the final event data readout hardware was installed at the ATLAS experimental area to allow for testing the complete physics event data readout chain.

The pre-series setup is used to validate the technology and implementation choices by comparing the final ATLAS readout requirements with the results of performance, functionality and stability studies. These results are also used to validate the simulations of the components and subsequently to model the full size ATLAS system.

Technical Runs and Deployment in Detector Commissioning Phases

Technical runs are held every one to two months and are interleaved with detector commissioning phases**.** Both are conducted on the final system and are run for a time period of one to two weeks. These activities are organized in the same way as the data taking will be conducted in the final experiment and are controlled from the final ATLAS control room. The work program and the detailed readout system configuration are prepared in advance. Collaborators take shifts and developers are on-call for help. Information about the run is kept in an electronic log book to keep colleagues informed about the current status. The TDAQ technical run includes the TDAQ hardware and aims at integrating software releases, new versions of components, performance studies of critical parts of the system or interfaces to external components like conditions database writing or event storage in the computer center. The functionality which is required for the next commissioning phase is verified where the software is then deployed in tests which involve the detector specific readout electronics.

Technical runs and commissioning runs give the collaborators the chance to get hands-on experience in running the complete system in conditions which are close to final. It provides an efficient feedback loop with prompt corrections and suggestions for enhancements. These activities also attract team members who are not normally resident at CERN and give a momentum to the work and to the people who have invested many years on the development of hardware and software.

ATLAS ONLINE SOFTWARE MANAGEMENT

The ATLAS TDAQ system includes currently 160 packages, 4000 source files written in C++, Java and Python with a total size of 60 Mbytes for over a million lines of code provided by 30 developers. The system is built each night on two platforms for the optimized and the debug versions, which take a total size of 2 GBytes. At the beginning of the project ten releases per year were built, now only three major releases per year. CVS [9] is used for source code version control, and CMT [10], an open source tool provided and maintained by the ATLAS collaboration, is used as a configuration management tool. Serialisation of the release building over a cluster of nodes and parallelism by executing several different targets in different threads in parallel enables the build process to terminate in 3-4 hours. A custom made script presents the build status of each package on a web page and allows the user to retrieve more detailed information. RPM [11] is used as package manager. The preferred memory debugger and performance profiler is VALGRIND [12]. Documentation for the application interfaces is automatically generated for each release.

Basic check targets are run automatically with the nightly builds for each component and for the integrated system. A check target for the integrated software is helpful for finding incompatibilities of modifications in libraries and process communication software. These regular checks ease tracing problems promptly while the details of the code modifications are still fresh.

Release testing is performed in two or more steps. First the infrastructure software is verified by its experts. Corrections and refinements are done and when confident, the token is passed onto the next sub-group for the testing of its applications. Once the software from all the sub-groups is integrated, the release is built with the tested software. If necessary, corrections are applied later in the form of software patches.

THE ALICE ONLINE SDP

The approach of the development of the Data Acquisition (DAQ) system of the ALICE experiment [13] at CERN was investigated. The ALICE detector and its collaboration are much smaller than ATLAS but the DAQ has high requirements on the performance of event data storage.

The core development group started off with a team of four very experienced developers who had been working together on data acquisition development and support for more than a decade. The common working habits, working language and common understanding, together with the technical knowledge and experience in the field was a given from the start. The team, who was based at CERN, grew one by one to about 10 members and the newcomers could be integrated smoothly. The team was asked early on to develop a DAQ system to be used immediately for related smaller fixed target experiments at CERN. This system can be regarded as a prototype featuring the main architectural lines of the final ALICE DAQ system notably in the sector of event building.

Given the exceptional composition of the team, the Domain Analysis and Brainstorming phases could be reduced to a minimum. Formal user requirements were written only for those parts of the system which were new to the team and for interface definitions to the hardware. Software had to be delivered in short time for the on-going experiments and therefore the software lifecycle was very short. Helped by the small size of the team and its location at a single place, the adopted SDP resembled the XP method. Common understanding of the technical issues and familiarity with colleagues made this method work. Unlike in XP, the team provided and maintained thorough user documentation from the very start. This was not only beneficial to the user but allowed team members to discover inconsistencies.

Currently, the ALICE central DAQ consists of O(10) packages and 120 000 lines of code. CVS is used as code management system and RPM as package manager. Fife to ten software releases per year are provided.

Two complete test systems are permanently available, one of them in the experimental area of ALICE. Commissioning with individual detectors is ongoing. Testing is emphasized in particular through the ALICE Data Challenges [14], where simulated detector data is moved from dummy data sources up to the recording media using as realistic processing elements and data-paths as possible. The nominal performance of 1 GB/s for recording data onto tape has been reached.

DO'S AND DONT'S

In the HEP environment, project managers as well as developers aim to spend as few resources as possible on a process which does not seem to be part of the final product. Compromises are made to realize a working system in time and according to the objectives. From the cases described above in the ATLAS TDAQ project and in the ALICE DAQ project, the most prominent factors to assure quality and success in developing a software system in the HEP world are drawn as follows:

- Give ample time and importance to building up a **project culture**. Common understanding of the project and its environment, a common working language, well defined terms and the use of external and in house standards are the basis for a fruitful development process. Means to include new team members should be exploited.
- Build **change management in** the SDP to give flexibility when requirements are modified and when the software environment is evolving.
- Define the functionality of a project at an early stage for maximum pay off in the development process. Misunderstandings and mistakes made at this stage when specifying the **requirements** make time consuming re-work of design and code necessary.
- **Review** the requirements thoroughly. Formal inspection is recommended for conceptually prominent components and interfaces to hardware and adjacent software projects. Light inspection can be applied once the principle and mechanisms of review are well accepted in the team.
- Make **testing a habit** and do it as **early** as possible and throughout the development phases. Resources and activities should be planned for prototype testing, release testing, unit testing, integration testing in small and in final scale, and for the support during commissioning tests.
- Use a **code management system**, **configuration management** and adopt **from the start** the concept of **planned releases** and **nightly builds** with **check targets** or an equivalent structure depending on the state of the art at the given moment of project development.

CONCLUSIONS

The SDP working environment of a large HEP experiment like ATLAS is challenging in terms of project size and life time, number of collaborating institutes and changing team members. Adapting a flexible SDP framework has helped the ATLAS TDAQ project to overcome those difficulties and to build up a working environment which eases communicating information amongst colleagues and taking up innovative ideas of the team members. The evolution of the SDP has followed the changing phases of the project over time. Current deployment in detector commissioning activities demonstrates the success of the team and of the approach.

ACKNOWLEDGEMENTS

My thanks go to the ATLAS TDAQ project members and its management for the fruitful collaboration we experienced during the many years of project building.

I thank Pierre Vande Vyvre from the ALICE experiment for kindly providing the information about the ALICE online SDP.

REFERENCES

[1] ATLAS Collaboration, "ATLAS Technical Proposal", CERN/LHHCC/94-43, LHCC/P2, CERN, Geneva, Switzerland, 1994
[2] B. Gorini et al, "The ATLAS Data acquisition and High-Level Trigger: concept, design and status", CHEP 2006, Mumbai, India
[3] K. Beck, "Extreme Programming Explained", 1999
[4] I. Alexandrov et al, "Impact of Software Review and Inspection", CHEP 2000, Padua, Italy
[5] Tom Gilb, Dorothy Graham, "Software Inspection", Addison Wesley Longman, Inc., 1993
[6] RuleChecker, http://www.itc.it/ ; http://atlas-computing.web.cern.ch/atlas-computing/projects/qa/tools/RuleChecker.php
[7] D. Burckhart-Chromek et al., "Testing on a large scale: Running the Atlas Data Acquisition and High Level Trigger software on 700 pc nodes", CHEP 2006, Mumbai, India
[8] N.G.Unel et al.,"Studies with the ATLAS Trigger and Data Acquisition 'pre-series' setup", CHEP 2006, Mumbai, India
[9] Concurrent Versions System, http://www.cern.ch/cvs
[10] The configuration management environment, http://www.cmtsite.org/
[11] Package Manager, http://www.rpm.org/
[12] A suite of tools for debugging and profiling Linux programs, [http://valgrind.org/]
[13] ALICE Technical Proposal for A Large Ion Collider Experiment at the CERN LHC", CERN/LHCC/95-71, 15 December 1995
[14] T.Anticic et al, "Challenging the challenge: handling data in the Gigabit/s range", CHEP2003, La Jolla, California