SOFTWARE EVOLUTION: CASE STUDY, OPAL

K. Ackerstaff
CERN, Geneva, Switzerland

Abstract
The OPAL case study presented in two lectures shows the application of modern software engineering techniques presented in the lectures on user requirements, project management, configuration management, software documentation and software quality in the Software Evolution Track in a running high energy physics experiment.

1. INTRODUCTION

In HEP (High Energy Physics) the use of computers is no longer restricted to numerical calculation of complex mathematical problems. Today computers and software are used in all areas of HEP experiments, from administration to physics analysis. The size and complexity of current and future HEP projects makes computers and software a major tool to perform fundamental physics research. Software has become a “mission critical” part of HEP and has to be treated with a professional approach. With the advent of large scale experiments involving more than 300 people, huge apparatus, a major financial investment and a lifetime of more than a decade, the management of people, money, hardware and software engineering becomes a problem which can only be solved using the expertise of professionals. Feedback from Software Engineering Science into HEP is essential to cope with problems in software evolution in HEP.

The application of modern software engineering technology and tools is not very widespread in current HEP experiments. The tradition, or culture, of software development in physics departments of universities and institutes is one reason for this. Traditionally software for physics has been produced by physicists and many of the foundations of software engineering actually stem from physics research. Meanwhile the use of computers and software outside research has increased dramatically and software produced for HEP is only a tiny fraction of the total. The introduction of software engineering methodology and tools into the HEP community is very slow. Since many of the developers are not software engineers, new methods need training and the reluctance of people to change their way of producing software must be overcome. The culture of software development in HEP, the “way we do things”, can only be changed gradually since the people involved mostly spend only a fraction of their time on software development and come from different institutions from all over the world. “A new way of doing things” can not just be imposed on developers, but the challenges of yet bigger collaborations and experiments, with more and more complex software and an increased dependency of the final outcome of HEP experiments on this software requires the use of modern software technology.

In this case study the introduction of modern software development techniques into a running HEP experiment is described.

2. THE OPAL CASE

The OPAL experiment is one of the four experiments at the LEP particle accelerator at CERN which has been in operation since 1989. Amongst other software and hardware upgrades, the online system was upgraded and partly redesigned in the period 1996-98. With the advent of the LEP2 era in 1996, the accelerator is being continuously upgraded to gradually reach a maximum centre-of-mass energy of approximately 200 GeV. The four experiments, which detect particle collisions at LEP, must adapt to the changing accelerator conditions. The OPAL experiment has experienced substantial increases in background. The planned increase in both the beam energy and intensity is expected to degrade background conditions further.

The OPAL trigger and data acquisition system must be able to handle this background without losing physics events. The original system contains considerable flexibility in programming the fast logic used in trigger decisions. However, during the (long) lifetime of the experiment the trigger control software has had to adapt to changes in the detector, the trigger hardware and the accelerator operating modes, which were not foreseen in the original design. This resulted in a control code which was excessively complex and difficult to maintain.

The difficulties experienced in adapting the trigger control software to these unforeseen changes motivated the redesign of the software. In turn, the positive experience from this redesign led to an upgrade of the event builder involving a redesign of all concerned software.

This case study is restricted to the trigger upgrade which enables to show all aspects of software evolution and use of modern software technology in OPAL. The redesign and implementation of the OPAL trigger software was carried out by a small team from the OPAL experiment in close collaboration with one member of the CERN/ECP Information and Programming Technology (IPT) group. The adoption of suitable methodologies ensured the successful outcome of the project, despite the heterogeneous and real-time environment and the strict time constraints of a running experiment.

3. THE OPAL TRIGGER

The OPAL trigger system [1] selects events according to logic implemented in several custom made VME compatible modules. Programmable combinations of 120 trigger signals from 32 subdetectors are used to form the trigger decision, made every 22 μs. A positive trigger decision is sent on a dedicated bus to the subdetector data acquisition systems. After complete readout the subdetector and trigger data are collected by the event builder. The complete events are sent through the filter [2] to data storage. The entire data acquisition system is controlled by RunControl using finite state machines (Further details of the OPAL data acquisition system are given in [3]).

The original software ran on two VME CPUs and controlled and monitored the trigger hardware and signals. It was written in real-time Fortran [4], consisted of approximately 60000 lines, and was implemented under the OS9 operating system.

The main tasks of the trigger software are:

- to load the user defined trigger logic into the hardware;
- to read out the trigger hardware on each event upon a positive trigger decision;
- to trigger and synchronize subdetector readout;
- to read out monitoring information asynchronously to the event loop and send the data across the network;
- to handle exceptions generated by hardware and software.

The user interface to the trigger monitoring was a commercial histogram presenter which ran on a Macintosh II [5] connected via a VME interface card to the trigger crate.

4. THE REDESIGN

A team of six people worked full- or part-time on the redesign of the trigger software, in a project which lasted from January until August 1997. The project was led by a physicist and supervised by the OPAL online coordinators. The redesign was split into three sub-projects: the core trigger code; the histogramming presenter; the trigger monitoring software. The redesign of the core trigger code is used here as an example to demonstrate the importance of the approach to software engineering.

4.1 Motivation

The experience of running the trigger system in 1996 when LEP2 started motivated the decision to redesign the software and streamline the hardware. The main motivations were to increase maintainability and flexibility.

Several hardware upgrades carried out in the past could not be fitted into the original design of the old code, which resulted in excessive code complexity. The documentation did not reflect many of these code changes. Further hardware upgrades are foreseen during the remaining lifetime of the experiment, and the manpower required for operations must be reduced. Maintainability can be improved by moving the non real-time functionality to a UNIX platform and thus simplifying the VME hardware. Moving the histogram presentation to the common UNIX platform also removes dependence on specific hardware, and reduces the number of operating systems used.

4.2 Requirements & Constraints

The new system should cover the full functionality of the old trigger system, and satisfy additional requirements in order to implement the improvements mentioned above.

- The core trigger and the monitoring code have to be separated. The remaining core trigger code has to run on a single VME CPU with no loss of performance.
- The monitoring code has to be moved outside the real-time system.
- The histogram presenter has to be replaced and run on a Unix platform.
- All software must be documented, and designed to be maintainable.
- The design has to facilitate future hardware changes.

To achieve this an object oriented approach was chosen, using the C++ programming language. The redesign of a central software system within a running experiment imposes several major constraints.

- The existing interfaces between the trigger and outside software systems must be retained, for example databases, event structures and RunControl interfaces.
- The new system must run with the old trigger hardware.
- To ensure a smooth transition, the new system must initially be backward compatible to avoid any loss of data during commissioning.
- Finally the new trigger system must be complete and operational for 1997 data taking, using the manpower available.

4.3 The Software Development Process

The first step in the project is the choice of methodologies in the different activities and phases shown in figure 1.

The user requirement document (URD) and the project management follow ESA's Software Engineering Standards (PSS05) insofar as was appropriate to the project.

For design and implementation the object modeling technique (OMT) was used.

Quality assurance was carried out by peer review and regular analysis of the code quality according to the ISO9126 [8] standards. The project was reviewed on a weekly basis in project meetings. The OPAL online and operations coordinators took part in the review process to ensure that requirements were met.

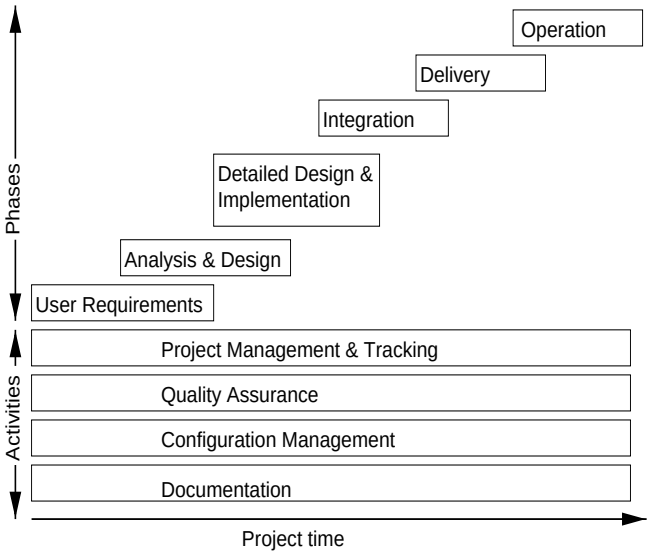

Fig. 1 The software life cycle used in this project. Activities such as documentation and management continue throughout the lifetime of the project, whereas project phases are active only during a specific time.

4.4 The Software Development Environment

In order to instrument the software development process a software development environment (SDE) was set up at the beginning of the project. Tools for each activity and phase of the process were selected and customized to the specific requirements of the project. Documentation is written using FrameMaker [9] with templates provided by the CERN/IPT group. These documents are hyperised using WebMaker [10] and linked to the OPAL online web. The source code is hyperised using LIGHT [11]. For configuration management the CVS [12] package is used. All relevant documents, utilities and source code are managed under the CVS repository, and changes to the repository are logged automatically on the web. CVSWeb [13] is used to browse the central repository via the web.

Code is developed on Unix workstations, whereas compilation, testing and operation take place on OS9 systems. For file and directory transfer across these platforms a tool has been written specifically for the project. The quality of the code has been analyzed along ISO9126 quality standards using Logiscope [14], which has been integrated to work with the GNU-make [15] program. Thus GNU-make allows compilation and linking on OS9 systems and automated quality analysis on Unix platforms using the same utility.

As an instrumentation of OMT, Rose [16] is used to produce the C++ class and inheritance as well as message trace and state diagrams.

The ATLAS coding conventions [17] were customized for the project. SNiFF+ [18] is used as the code development environment.

The knowledge and experience of the software engineer from the IPT group has been essential in setting up the SDE and choosing appropriate methodologies and tools. The project team benefited from this in terms of learning and applying the methodologies and tools. The setup of the SDE and the integration of the tools described above took approximately a month at the start of the project. Analysis, design and implementation took altogether three months, and the integration and testing phase two months. This time could only partly be used due to the constraints of the commissioning of the OPAL experiment and the LEP schedule, when the OPAL data acquisition system was often unavailable for testing the trigger.

4.5 Experience with the Software Development Environment

A detailed documentation of the project infrastructure includes a description of all technical actions in the software development process. This allowed all members of the project team to rapidly learn standard actions and adapt to the SDE. Therefore all tools could be used from the outset of the project. This documentation proved to be absolutely essential in order to get people to use and work within the SDE.

A substantial amount of information on the details of the trigger system, for example exception handling and interfaces, had to be retrieved from the old trigger code. SNiFF+ has been used to analyze the old software, and FrameMaker used to document the software and hardware, which facilitated the re-engineering process. A detailed quality analysis using Logiscope helped determine the critical parts of the trigger software. SNiFF+ also allows simultaneous navigation through both the old and new software during the analysis phase.

The OMT design enables fast and efficient coding. The models produced with Rose can be tested against the functionality of the old code. Hence the implementation of the models is separated from design and can be carried out by any member of the project team. Furthermore, inheritance allows partial testing. For example, the trigger hardware is reflected in hardware classes inherited from a VME module base class. This base class hides all VME specific memory access and error handling. This class allows either direct access to hardware registers, or else the registers are simulated in local memory if desired. This allows the software to be run on a test setup with only a subset of the hardware modules present. Hardware which cannot be accessed is automatically simulated in software.

The members of the project team were able to work simultaneously on the code using CVS. Following the coding conventions results in a uniform code structure which facilitates peer review and collaborative coding. The majority of version conflicts are resolved by the CVS merging feature. Version tracking and logging are important also during the implementation phase.

4.6 Product Assessment

Figure 2 shows the overall quality of the new code against time compared to the old code. The percentage of code rated as fair or poor according to the ISO9126 standards is below 2%, compared to above 30% for the old code. The improved code quality is maintained throughout the implementation and integration phases.

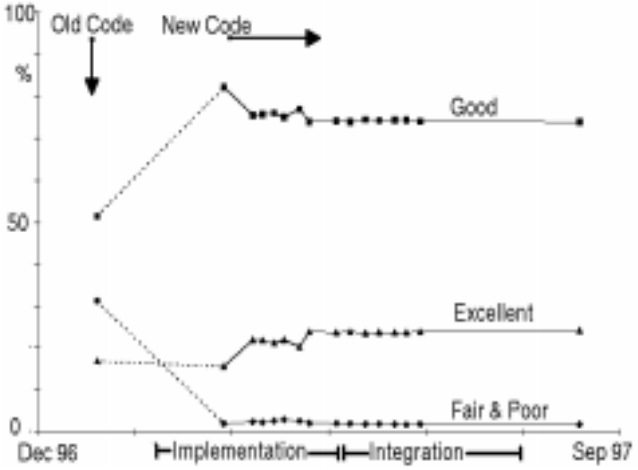

Fig. 2 ISO9126 quality of the old and new code. The percentage of the code in three quality categories is shown. The values for the new code are shown over the duration of the project.

The improved quality is reflected in the performance of the new code. It has been operational since the beginning of August 1997, and satisfies all requirements. Unlike the old system no expert intervention has been required during data taking.

The control cycle and exception handler synchronization is carried out by OS9 signals rather than by introducing artificial delays, as was the case beforehand. As a consequence the software is much more stable. A recent hardware upgrade and associated software changes were completed in one afternoon rather than several weeks needed for the previous upgrade. This is due to the flexibility inherent in the design. Moving the monitoring software outside the real-time system protects the core control code from frequent intrusion. Monitoring tasks are automated and data driven from configuration files, thus allowing changes by the user without modifying the code. Due to this and the full documentation maintainability is improved.

5. CONCLUSIONS

The OPAL trigger software has been redesigned. The project was completed on schedule and met all requirements, despite the constraints imposed by the timelines and existing interfaces of a running experiment. The successful outcome is a direct consequence of using modern software engineering methodologies.

The customization of the methodologies and tools together with a cookbook like documentation on how to use the SDE makes it easy for new developers to adopt the environment and work with it. The process of this customization carried out by a software engineer and physicists exhibited the difference in their “culture”. Carefully adapting methodologies and tools to the specifics of the project is absolutely necessary. The PSS-05 standards for User Requirements for example are much too detailed for a project like this, nevertheless after customization they proved to be very useful in capturing the user requirements.

The SDE established for the trigger upgrade project now forms the basis for the largest part of the OPAL online software. The redesign of the OPAL event builder made use of it and existing software has been incorporated into the same environment. This did not happen because of a management decision, but because it proved to be useful to the maintainers and developers of the software.

In general the project profited from the methodology and the management techniques, the tools as the instrumentation of the SDE are secondary in that they can be exchanged by similar tools from

different suppliers. The change in the culture, the “way we usually do things” in the HEP community, is the major point to be achieved to improve software engineering and software quality in HEP experiments.

ACKNOWLEDGMENTS

The successful completion of the project would not have been possible without the active support of the OPAL online coordinators, Per Scharff-Hansen and Frans Meijers, the OPAL trigger coordinators Graham Wilson, and Tara Shears. The help of a professional software engineer, Arash Khodabandeh, with the know how of the CERN/EP/IPT group was absolutely essential for this work, we find it important to have a group acting as internal consultant in CERN to guide us in software engineering matters. In addition we wish to thank Denice Deatrich for writing the new histogram display, and Frans Meijers and Christoph Schwick for their contribution to the trigger monitoring. We gratefully acknowledge the financial support of the Particle Physics and Astronomy Research Council, U.K.

REFERENCES

[1] M.Arignon et al. “The trigger system of the OPAL experiment at LEP,” Nucl. Instr. and Meth. A313 (1992), pp. 103-125.

[2] D.G.Charlton et al. “The on-line event filter of the OPAL experiment at LEP,” Nucl. Instr. and Meth. A325 (1993), pp. 129-141.

[3] J.T.M.Baines et al. “The data acquisition system of the OPAL detector at LEP,” Nucl. Instr. and Meth. A325 (1993) pp. 271-293.

[4] H. von der Schmitt, “Real Time Fortran”, available from the OPAL Secretariat, CERN, 1211 Geneva, Switzerland.

[5] Macintosh II and AppleTalk are trademarks of Apple Computer Inc., Cupertino, CA 95014, California, USA.

[6] “Software Engineering Guides”, C.Mazza et al. Prentice Hall, London 1996.

[7] “Object Oriented Modeling and Design”, J. Rumbaugh et al., Prentice Hall 1991.

[8] ISO/IEC 9126, “Information technology - Software product evaluation - Quality characteristics and guidelines for their use,” International Organization for Standardization and International Electrotechnical Commission, December 1991.

[9] FrameMaker, available from Adobe Systems, Adobe House, West One Business Park, 5 Mid New Cultins, Edinburgh EH11 4DU, Scotland, United Kingdom.

[10] WebMaker, available from Harlequin Limited, Barrington Hall, Barrington, Cambridge (GB), CB2 5RG.

[11] LIGHT, available from CERN/ECP/IT, http://www.cern.ch/LIGHT/.

[12] “Version management with CVS, CVS 1.9”, Per Cederqvist et al., available from Signum Support SA, Box 2044, S-580 02 Linkoping, Sweden.

[13] CVSWeb, a utility to browse CVS trees via the web, see http://www.freebsd.org/~fenner/cvsweb/

[14] Logiscope, available from Verilog SA, 150 rue Nicolas Vauquelin, P.O. Box 1310, F-31106 Toulouse Cedex, France.

[15] “GNU make, A program for directing recompilation, edition 0.5 for V. 3.75 Beta” R. M. Stallmann and R. McGrath, Free Software Foundation 1996.

[16] Rose, available from Rational Software Corporation, Immeuble de la gare, 1,place Charles de Gaulle, f-78180 Montigny le Bretonneux, France.

[17] “C++ coding standards for ATLAS”, S. M. Fisher and L. Tuura, CERN Atlas Software Note 29 (1996). http://atlasinfo.cern.ch/Atlas/documentation/notes/SOFTWARE/note29/cxx-rules.html

[18] SNiFF+, available from TakeFive Software GmbH, Jakob-Haringer-Strasse 8, A-5020 Salzburg, Austria.