Computing and Software for Big Science (2021) 5:6
https://doi.org/10.1007/s41781-020-00046-8

EDITORIAL

Education and Training for Software Developers in Particle Physics

Stefan Kluth[1]

Particle physics emerged as a field starting with the early balloon experiments in the '30s of the last century. Universal digital computers were invented at about the same time, by Zuse and his company in Germany in 1941, and as the ENIAC [Electronic Numerical Integrator and Computer] machine in the United States in 1945.[1]

Nuclear and particle physicists were quick in realizing the potential of digital computers for the analysis of their analog data consisting of cloud or bubble chamber photographs of collisions produced in accelerator experiments or emulsions exposed to cosmic radiation in the atmosphere. The key to success was digitization of the images and subsequent statistical analysis of the data. Most of the basic pattern recognition and parameter extraction algorithms of our field were already invented in the 1950s and 1960s.

How is this relevant to our subject? The early adoption of digital computers, compared to many other scientific fields, led to the development of a separate "software culture" in our field. The basic common denominator with industry and other science fields was programming languages, mostly Fortran, and the operating systems. Up until the 1990s, and to some degree still continuing today, many tools and libraries were developed by physicists in the field. The young scientists had essentially no structured and pedagogical teaching about software development.

It is still an established pattern to assign full-time software development tasks to students or young postdocs with no training except studying some existing code. In this context, the development of a physics reconstruction, simulation or analysis workflow is mainly a software development task, and most of the time on the project is spent on developing, testing, debugging, and optimizing the code producing the physics results.

Few in our field would disagree with the statement that the software stacks of the big collaborations are not in a very good state. Core parts were developed decades ago, with the limited version of C++ of the time, and with sometimes poorly understood and implemented concepts of object-oriented software engineering. Some packages are now without consistent and knowledgeable maintenance, because it is difficult to find people ready to take responsibility for such legacy code.

Our subject here is education and training (Bildung und Ausbildung in German). The distinction between the two terms is important: training refers to the transfer of a particular technical skill such as understanding the C++ syntax, using libraries for parallel programming, or applying a debugging or performance optimization tool. Education goes much further, and should enable the student to understand concepts of software engineering and computing transcending programming languages, specific libraries or details of the current hardware.

We can work successfully to improve the state of our software not only in terms of run time or memory efficiency, but also in terms of ease of maintenance, simplicity of adding new features, and absence of errors. In order to achieve this we must focus on the people doing the work and give them the training and education they need to solve their tasks with confidence and professionalism.

I was lucky to benefit from education on object-oriented software engineering as a postdoc at the Lawrence Berkeley National Laboratory (LBNL), Berkeley, California in 1998. At this time the BaBar Collaboration organized these courses in order to boost the development of the experiment software in C++. The instructor was Robert C. (Uncle Bob) Martin, today one of the best-known software engineers, co-author of the Agile Manifesto in 2001, and author of numerous popular books on software engineering. I realized later, working again on the OPAL experiment with its old Fortran code base and then on ATLAS, how much I had benefited from the education in Bob Martin's courses, since I could now see the concepts and strategies behind such large software projects.

✉ Stefan Kluth
skluth@mpp.mpg.de

[1] Max Planck Institute for Physics, Munich, Germany

[1] The Turing-completeness of the Zuse Z3 was shown only in 1998 and was not a design goal of its inventors.

Published online: 03 February 2021

I decided to pass on this knowledge and started to develop my own courses, merging concrete examples from particle physics with object-oriented software engineering concepts. The courses were held at the Max Planck Institute for Physics, for the ATLAS collaboration at CERN, as lectures at the Ludwig-Maximilians-University (LMU), or as contributions to software and computing schools. Later, collaborating with my colleagues, in 2010 we began a series of "Advanced programming concepts" workshops, adding the topics of "unit testing", "refactoring", performance optimization, and generic programming to the program [1]. The evaluation of feedback from participants [1] was generally rather positive and encouraged us to continue the effort.

There are of course many schools on software and computing aspects in particle physics and related fields, such as the CERN School of Computing (CSC), the GridKa School, and many others (see e.g. [2]). These events cover a broad range of important technical aspects of software development and computing systems, and play an important role in providing training in our community. However, none of these events will help to fundamentally improve the suboptimal quality of a large fraction of our software, since this would require not only technical improvements but also a general appreciation of the main driving forces in a large project by at least a majority of the contributors.

Improving on the state of education of our developers would require stepping back from teaching technical aspects in some cases and making room for the big concepts like object orientation, generic programming, dependency control, refactoring, and unit testing. A well-educated developer will be able to discover the need for a certain technical skill by herself, but even the technically excellent programmer, after visiting many schools without a good grasp of the concepts, may find himself wasting time and effort on unessential problems.

Now, I have made some perhaps provocative statements, and not all readers will agree. The problem is that we have very little hard information publicly available, such as software quality metrics, unit test coverage, or studies of efficiency of development workflows, to name just some examples of interesting topics.

In order to change this in the future I would like to encourage you to study such aspects of software development in your domain and publish the results in this journal. If you are involved in teaching topics in software development and computing, as training or education, please collect feedback from participants and publish your findings here. Such reports will help us to identify problems with software development in our community and find ways to address them by providing adequate training and education.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Kluth S, Pia MG, Schoerner-Sadenius T, Steinbach P (2015) "How do particle physicists learn the programming concepts they need?. In: CHEP 2015 proceedings, arXiv:1505.04604
2. hepsoftwarefoundation.org/Schools/events.html