Making Modern Scientific Software Development Explicitly Agile

Magnus Sletholt
University of Oslo
magnusts@ifi.uio.no

Hans Petter Langtangen
Simula Research Laboratory
and University of Oslo
hpl@simula.no

Jo Erskine Hannay
Simula Research Laboratory
johannay@simula.no

Dietmar Pfahl
University of Lund
dietmar.pfahl@cs.lth.se

The nature of scientific research and the development of scientific software have similarities with processes that follow the agile manifesto: responsiveness to change and collaboration are of the utmost importance. But how well do current scientific software development processes match the practices found in agile development methods? As a representative for a modern scientific software project, we briefly outline the FEniCS project whose objective is to automate solution of differential equations. Based on initial investigations, we propose a case-study where we investigate how the development process in FEniCS can be mapped to agile development methods, and whether SCRUM can be suitable to administer such a project. The case study will use another project, the more traditional Dalton project, as a contrast.

Introduction

Scientific software operates in a variety of domains and development, as well as use. According to D. Kelly, “a chasm opened between the scientific-computing community and the software engineering community” [1]. Her study suggests that these domain-specific fields of science may be a contributing factor as to why just a small fraction of research in software engineering is oriented toward scientific computing. Software engineering research has traditionally focused on techniques, methods and concepts that can be made available in an array of domains.

Scientists use their software, and the results of executing the software, to do complex calculations or simulations. In some scientific projects, the software may be used in order to test a scientific theory. These characteristics of scientific software entail that, in contrast to the development of, say, administrative or business enterprise software, the writers of scientific software cannot determine what the correct output of an application should be in the traditional sense. This poses two particular challenges from the software engineering point of view: First, requirements elicitation and specification will be highly dynamic. Due to the exploratory nature of many scientific projects, the elicitation and specification of requirements is problematic because they may be unclear, or even unknown, up-front. In fact, to the degree that any specification is perceived necessary, requirements are written near the completion of the software. The requirement activity is a recurring problem in scientific software projects, especially when the teams are large or when scientists dedicate much time to developing software [2]. Secondly, the definition of test cases for validation and verification of the software is extremely challenging. It is often not obvious to stipulate

whether an error lies within the scientific theory or in the implementation of that very theory. Although these aspects of scientific software impede requirements handling and testing in the outset, the lack of knowledge about requirements and testing principles and the lack of organized activities have been identified as a problem area in several studies [3; 4; 2].

According to [4], documentation is important amongst scientists that develop software, primarily documentation concerning the scientific theory. Other documentation, such as software-technical documentation and user documentation, is not as emphasized. These kinds of documentation are very rare, sometimes non-existing, in many projects, although there often is a partial overlap with the theory documentation.

In most aspects of the development of scientific software, the urge to conduct science is the primary motivation and goal. Scientists therefore have a different approach to developing software than software engineers; their mindset is to perform science, not to write software [5]. The variation in domains and motivation found in scientific software projects are naturally factors that need to be taken into account when choosing work and development methods. The development method of choice is usually, informally, the “best method” [3], and there exist huge differences both across and within the domains as well.

Mapping Scientific Software Development to Agile Methods

Long development-cycles are common in scientific software projects, where the piece of software is the culmination of the combined effort performed by a number of scientists over the course of many years [6]. Due to this development model where modules are individually added to the application, and due to the challenges with determining requirements up-front, the processes tend to be more agile-oriented than process-heavy [3]. Sanders supports this notion by stating that most projects under investigation in her study had an iterative, rather than a plan-oriented, approach to development [4].

The FEniCS Project

FEniCS is a joint project between University of Chicago, Argonne National Laboratory, Delft University of Technology, Royal Institute of Technology KTH, Simula Research Laboratory, Texas Tech University, and University of Cambridge. The vision of FEniCS is to set a new standard in Computational Mathematical Modeling (CMM), which is the Automation of CMM (ACMM), towards the goals of generality, efficiency, and simplicity, concerning mathematical methodology, implementation, and application.

The software developed in the FEniCS project consists of a dozen software components, written in C++ and Python. The project members and an international user community participate in the development of FEniCS components, applications, and documentation.

Most FEniCS simulators are written in Python, but the Python program generates C++ code tailored to the problem at hand, and links this C++ code to general libraries for finite element computations, linear algebra packages, etc. Simula Research Laboratory has the primary responsibility for distributing FEniCS as an open source system. Building and testing FEniCS components with and all its dependencies, such as PETSc (a portable, extensible toolkit for scientific computation) and Trilinos (an object-oriented software framework for the solution

of large-scale, complex multi-physics engineering and scientific problems), can quickly become a nightmare. To tackle these problems successfully, Simula Research Laboratory has a dedicated scientific programmer working with a Buildbot system for FEniCS as well as packaging FEniCS for Debian and other binary distribution repositories.

Investigating Agile Practices in the FEniCS Project

Among the most salient agile features of the development process in FEniCS is the high degree of self-organisation within the sub-projects and their teams, and the focus on providing working software in short iterations (instead of having a long implementation phase before anything presentable is available).

A deviation to agile principles is the way developers in FEniCS typically communicate with each other. Typical communication channels are public mailing lists and the version control tool Launchpad where developers from an international community can check out and commit code. Launchpad also provides functionality for bug tracking and issue reporting. Developers can create and edit “blueprints”, which is a description or specification of a new feature or the alteration of some existing functionality. On the other hand, at one of the major developing partner sites of FEniCS, Simula Research Laboratory, there is frequent informal communication taking place among co-located scientists involved in the development of software for FEniCS.

In FEniCS, the individual developer usually selects a task voluntarily driven by own needs and without much discussion or external pressure. Developers are free to solve their tasks in the way they find fit. This also applies to testing activities. However, as there is no particular focus on testing and no guidance as to what kind of testing is required, the tests are of variable quality if they exist at all.

Once a scientist assumes a task, this task is usually not further distributed or delegated, with the exception, perhaps, when a scientist assigns the implementation of a feature to a PhD student.

In FEniCS, as in many scientific software development projects, the customer and the developer are often the same person, as the purpose of picking a development task is often to support one’s own research. It is, however, certain that the developers who contribute to FEniCS also are the users of the software they develop. This deviates from a developer-customer collaboration pattern in the traditional sense, but it is evident that this overlap of interest means that the developers are highly motivated.

Case Study

There are fragments of agile elements present in the FEniCS project. There is also, among some project members, a desire to have a more defined development process. We will therefore conduct an empirical case study [7] on the FEniCS. The purpose of such a case study is two-fold: to analyze and conceptualize the core process elements of the software development processes in the FEniCS project, and to see to what extend these process elements map to core agile techniques and management practices (e.g., Scrum). The case study will be exploratory in the first part and confirmatory in the second part. Research

methods will include techniques for eliciting and externalizing practitioners' tacit knowledge [8; 9; 10].

The FEniCS project is one of largest projects of its kind, and therefore gives a rich observational base for a case study with our purpose. The case study will not claim generalization to other scientific software development efforts per se; it will first and foremost be a proof of concept study. However, generalization is possible to the extent one is able to identify process elements in other projects similar to those in FEniCS.

In order to further delineate potential agile process elements in the FEniCS project, a contrast project will be studied in parallel. The Dalton project represents a more traditional approach to scientific software development. The general domain of the program is quantum chemistry, with an emphasis on molecular electronic structures. The aim of the program is to automate the determination of such molecular properties. Seven separate components, with more or less separate development cycles, form the complete program Dalton. Since the beginning of the project, many scientists from an international community have contributed to the development. The first version of Dalton (1.0) dates back to 1997. There were two additional releases in the subsequent years, leading to the current version 2.0, which was released in 2005. Between these versions, minor patches were issued in order to correct programming errors. FORTRAN.77 and C are the programming languages used in Dalton. The program is easily installed using a supplied makefile script on a UNIX platform. There exists a test suite that the user can run in order to ensure that the program was installed successfully.

Conclusion

Following earlier and ongoing investigations into the practices of scientific software development, we propose that it is valuable to further analyze and conceptualize the core process elements of such development, in terms of modern software engineering best practices. A likely outcome of such investigations will be more explicit and deliberate scientific software development practices, and also a timely updating of software engineering methodology to include domains other than business-administrative software.

References

[1] **Kelly, D.F.** *A Software Chasm: Software Engineering and Scientific Computing.* 2007.
[2] **Hannay, J.E., et al.** *How Do Scientists Develop and Use Scientific Software?* 2009.
[3] **Carver, J.C., et al.** *Software Development Environments for Scientific and Engineering Software: A Series of Case Studies.* 2007.
[4] **Sanders, R.** *The Development and Use of Scientific Software.* 2008.
[5] **Decyk, V.K., Norton, C.D. and Gardner, H.J.** *Why Fortran?* 2007.
[6] **Sanders, R. and Kelly, D.** *Dealing with Risk in Scientific Software Development.* 2008.
[7] **Yin, R.K.** *Case Study Research: Design and Methods.* Sage Publications, 2003.
[8] **Jarvis, P.** *The Practitioner-Researcher.* Jossey-Bass Publishers, 1999.
[9] **Argyris, C.** *Knowledge for Action.* Jossey-Bass Publishers, 1993.
[10] **Argyris, C. and D.A.Schon.** *Organizational Learning II. Theory, Method, and Practice.* Addison-Wesley Publishing Company, 1996.