Adv. Geosci., 45, 383–387, 2018
https://doi.org/10.5194/adgeo-45-383-2018

Advances in Geosciences
Open Access
EGU

Opportunities and limitations of software project management in geoscience and climate modelling

Nadine Wieters and Bernadette Fritzsch

Alfred-Wegener-Institut Helmholtz-Zentrum für Polar- und Meeresforschung, Bremerhaven, Germany

Correspondence: Nadine Wieters (nadine.wieters@awi.de)

Received: 30 June 2018 – Revised: 26 November 2018 – Accepted: 5 December 2018 – Published: 20 December 2018

Abstract. In this paper, recent software project management tools and their possible advantages, disadvantages, and possible limitations will be discussed, with respect to their application in scientific projects in geoscience and climate science.

1 Introduction

During the last decades, the general process of software development has strongly improved. In the industry, software engineering methods like version control systems and issue tracking are now indispensable to the everyday work of a software developer as well as agile and test driven software development and continuous integration. Somewhat different is the situation in science. Here, modern methods of software development have not yet been introduced everywhere, whereby the various scientific disciplines have a different status. Even within the disciplines, the knowledge about methods and tools is very different. For example, geoscientists already have advanced areas (such as geoinformatics) and some of the complex model codes are successfully managed today using software engineering methods (e.g. FESOM, Wang et al., 2014; ICON, Giorgetta et al., 2018). But from our experience, in too many other projects, too little attention is paid to software development practise. Therefore, in this paper, we especially want to make project managers aware of some aspects of software development and show how modern methods can help to successfully make software development a part of the project.

2 Software development in the scientific context

Scientific software and its development is of great importance in scientific research (Hannay et al., 2009; Hettrick et al., 2015). In geoscience as well as in climate science, research software (in the form of developed program code) is an integral part or even the basis of research in many fields and in many projects.

As the complexity of such research software and program code is increasing, also the development of the program code is getting more complex, especially when developed and used in larger interdisciplinary groups. One way to deal with these issues is to use software engineering concepts and tools which can help to improve the whole process of software development and software project management. Although there are software engineering tools that are already well established in general software development, such concepts and tools are underused in research software development so far. The question arises, whether recent software engineering tools and developing concepts can also improve the work on research software in geoscience. Are such tools and concepts as applicable as in general software projects? Or are research software projects different in the sense that they hold certain characteristics, which makes it difficult or even impossible to establish common software engineering tools and concepts?

Hannay et al. (2009) did a survey to find out how scientists develop and use scientific software. Some of the main findings that are relevant for our study can be summarized as follows:

- 84.3 % of the scientists state that the development of scientific software is important for their research;
- 96.6 % state that their knowledge about software development comes from self studies;

Published by Copernicus Publications on behalf of the European Geosciences Union.

- a notable number of scientists are not familiar with standard software engineering concepts;
- the importance of software engineering concepts is rated as high, especially in large projects.

In a more recent study, Johanson and Hasselbring (2018) have investigated key characteristics of scientific software development in computational science. They found a number of reasons for the low prevalence of software engineering methods in the scientific field:

- software requirements are not known up front;
- scientific software in itself has no value but still it is long-lived;
- little code re-use;
- only few scientists are trained in software engineering;
- disregard of most modern software engineering methods;
- overly formal software processes restrict research.

In our view, one key point is that most research software developers are not formally trained in software development methods and so, they are often not aware of recent software engineering tools and concepts. In addition, after our experience in geoscience, software development in projects is usually just a means to achieve the project's goal and not the goal itself. This sometimes leads to software development not getting the necessary importance in the project planning and that the necessary methods are not considered as crucial. This may lead to scarce incentives for high quality software development (unfortunately including documentation). Since the development process is not seen as a crucial task, the general work of software development is sometimes lacking in acknowledgement. Every effort that is spend in software management tools is rated as extra work and not be considered as an improvement of the development process.

Increasing lifecycles of climate models and data analysing tools, beyond the lifetime of computers, as well as increasing complexity of the overall development process require new development strategies. They must ensure that the developed software remains maintainable even after the project has finished with reasonable effort. Otherwise, there is a risk that the following projects that build on this software, have to deal repeatedly with maintenance procedures that require additional resources and hinder their project progress. In addition, development tools can add value by helping scientists to ensure reproducibility. Scientific work is fundamentally based on the fact that the results obtained can be reliably reproduced. For computational science this means that the computer simulations can be repeated using the identical code. With ever-evolving complex software with many lines of code, identifying the exact version for a numerical experiment can become complicated. Here, the version control, which will be discussed in more detail in the following, can help.

In the following section, we will present two software management tools that are widely used in general software development. We will point out how these tools can improve the above stated characteristics of research software development and what may be the hurdles of its establishment in geoscience.

3 Software project management in research software development

Driven by the experience of major software projects in the industry, software engineering has developed a variety of methods with different objectives. Among other things, they should ensure the quality of the created software as well as its maintainability throughout the entire lifecycle, enhance its reusability and reduce the time of delivery. They improve the work and coordination in distributed developer teams. In the following section, we will discuss two examples of software engineering tools: (i) version control systems (VCS), as a software development tool, and (ii) agile software development, as an example for a more strategic development method. Taking these examples, we describe how such tools can be integrated in the development process of scientific software development in geosciences and climate sciences.

3.1 Version control systems

To a large extent, research software is developed not only by individual scientists, but is the common task of a larger team of developers; some of which are working at the same time at different locations. For such a collaborative work, every developer must always know exactly what changes were made by the other collaborators and that they might influence his or her work. Version control systems (VCS) manage complex projects with many contributors. They track different states of a software and always support the restoration of defined versions of the software. The scientist can repeat the numerical experiment or the data evaluation with exactly the same software version and thus prove the reproducibility of the results. VCS are therefore not only beneficial for collaborating teams, but also for the individual developer, since they improve the collaboration with the future self (Bowers and Voors, 2016).

First version control systems have been developed in the 1970s, where they initially helped with the administration of the Unix system by protocolling changes in configuration files etc. (Rochkind, 1975) and developing operational systems. Later its field of application extended to every kind of files and documents and especially to software code. They fulfil several tasks. Changes are logged so that it is possible to understand who did what and when. Previous states can be

restored so that accidental changes or unsuccessful attempts can be reversed. All states are archived. This makes it possible to access all versions. Collaborative and shared access to the files from multiple developers will be coordinated and it will immediately alert to conflicting actions. Modern version control systems also support the simultaneous work on several development branches of a project and thus allow simultaneous progress in various subtasks.

When a model is co-developed by several scientists, a VCS is essential to make coding efficient and to avoid conflict through concurrent and conflicting changes. However, this often requires a fundamental change in the individual workflow of the developing scientists. Changes to the code must be checked in continuously for the system to perform its function. In some cases, the fact that the work of each developer is visible, also in unready stages, may mean a psychological hurdle that must first be overcome. Besides the issues that can be improved by VCS, as stated above, it is important to establish best practices how a VCS is integrated in the process of development. The development team must agree together on appropriate ways of working with a VCS. Only then, the opportunities of this tool can be fully exploited and the development can be improved.

For the scientific sector, open source solutions of version control systems play an important role, especially *Git* (https://git-scm.com, last access: 13 December 2018) and the older *Subversion* (SVN) (Fitzpatrick et al., 2004). In particular, *Git* has become widely used in recent years, as repository hosting services like *GitHub* (https://github.com, last access: 13 December 2018) or *GitLab* (https://about.gitlab.com, last access: 13 December 2018) have become more user-friendly and offer many additional features (e.g. issue tracking, wiki, continuous integration) besides pure code management.

3.2 Agile software development

A mainly linear approach of software development, like the so-called waterfall model, is based on discrete phases of analysis, design, coding and testing. The phases build on each other and are carried out in a predetermined sequence. Characteristic of this approach is the consistent implementation of the previously planned steps. Thus it has a high level of planning security. However, it carries the risk that later changes in the requirements can be taken into account only with great difficulty or huge effort.

As already mentioned in Sect. 2, the software requirements in scientific projects are often unclear at the start. This entails risks in the implementation of a waterfall model. Since the requirements for the software change, according to the iterative procedure that is usual in the scientific context, an agile procedure is appropriate (see also Easterbrook and Johns, 2009), which allows adapting quickly to changes. Many agile methods and practices have evolved from the foundations of the *Manifesto for Agile Software Development* (http://agilemanifesto.org/, last access: 13 December 2018). They are characterized by a continuous comparison of the expectations to the actual state of the software under development. Thus, they offer a higher flexibility. A prominent representative of the agile software development is *Scrum* (https://www.scrum.org, last access: 13 December 2018), which will be briefly described in the following.

The properties of the software are formulated from the user's point of view and recorded and prioritized in a *Product Backlog* by the *Product Owner*. These requirements are gradually being implemented by the *Development Team* in relatively short time intervals (so-called *Sprints*). The *Development Team* is responsible for the delivery of the functionalities in the order desired by the *Product Owner*. It organizes itself and decides how it works. At the end of each *Sprint* there is a finished product (*Product Increment*). At the end of a cycle, the product, the (possibly changed) requirements and the procedure are reviewed and further developed in the next *Sprint*. The *Scrum Master* is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables and ensures that the Scrum framework is followed.

The application of agility in a scientific project can take place in different ways. On the one hand, agile management often lends itself to the overall project, in which software development is included as an integral part. Then, software development is managed by the same principles as the other parts of the project. On the other hand, an agile approach in software development can be embedded as well in other methods of overall project management. In this case, it is important that the project manager understands the specifics of the agile method and takes them into account in the planning of the entire project.

4 Implications for research project management

In the following section we will give some ideas concerning software development that can be considered in project management so that the above described problems can be addressed already in the early phase of project development. We will formulate these ideas as questions that arise for the research software itself, infrastructure and personnel.

Regarding the research software the following questions arise:

- Which research software is needed to achieve the project goals?
- Are the features of the needed research software known already and is it clear to all project members?
- Is it necessary to develop new program code or can existing research software be re-used or adapted?
- Is the development process of program code in the project considered high enough, with enough resources in time and personnel?

– Are software engineering methods considered to be applied in the project?

The following questions are regarding resources and support for software as well as computational resources:

– Is there an infrastructure that can or should be used regarding certain tools, that have been agreed upon using? E.g. which version control system is in use?

– Are all the tools that are necessary available on this system?

– Are other tools needed, e.g. issue tracker, repository manager, wikis (e.g. for documentation)? And is it possible to install such new tools on the system?

– Is there support available for such tools, e.g. from the computing centre?

– Are there enough computational resources available?

Regarding personnel we would like to point to the following issues:

– Is there a special need for certain skills, that people need to have?

– Is it necessary to train personnel with these skills in the project?

– Are people needed, that have experience in software engineering?

– Is the knowledge about the research software and its development spread over a group of people, or is it centred in one person?

– Is it possible and necessary to enable and foster communication across different cultures of software engineering and computational science?

5 Ongoing activities

During the last years, the awareness for the role of research software and its importance in the scientific process has been increased, and was finally summarized in the slogan *Better software – better research* (Goble, 2014) which is used by the *Software Sustainability Institute* (https://www.software.ac.uk/, last access: 13 December 2018) and the *Research Software Engineers Association* (https://rse.ac.uk/, last access: 13 December 2018) to point out the significance of high quality software for science. Starting in the UK, people in academia with expertise in programming and intricate understanding of research, the so called *Research Software Engineers* (RSE), are now in the process of accumulating and community building. Many activities are ongoing to found national chapters, e.g. in Germany (https://www.de-rse.org/de/index.html, last access: 13 December 2018). Since the quality of the software depends crucially on the existing knowledge and skills of its developers, there is a great need for further education of the involved people. With the *Software Carpentry* (https://software-carpentry.org, last access: 13 December 2018; Wilson, 2016), there is now an extensive network of experts who share their experience and expertise in programming and best practices.

Furthermore, several national initiatives address research software. In Germany the Helmholtz Association implemented a task group *Access to and re-use of scientific software* to enforce the discussions about scientific software in the Open Science context. The task group formulated guidelines and recommendations for development and publication of scientific software that have also been presented to the geoscience community (Fritzsch et al., 2017). On the level of the Alliance of Science Organisations in Germany, the Priority Initiative *Digital Information* has a working group *Research Software* to work out a coordinated position of all German Science Organisations. First outcome of this working group has been published in Katerbow and Feulner (2018).

6 Conclusions

We described some aspects of software development in geoscience projects. From this, some conclusions can be deduced. Even if new software is not the primary goal of a project, the project management has to consider the software development as an integral part of the project from the very beginning. In Sect. 4, we therefore proposed a collection of questions and considerations that can be addressed in the early phase of project management, to ensure that the software development is taken into account in the overall project planning. The development of high-quality software needs time and personnel resources. Software management tools can support and improve the development of scientific software, but its integration in project teams needs coordination and consensus in its way of application. This selection and integration process needs to be considered as part of the software development within a project. To cope with a lack of awareness and knowledge in software engineering tools high focus should be taken on training and considered to be part of the research project. Within the project and beyond, the exchange of knowledge and experience between geoscientists and software engineers should be promoted to overcome communication and cultural hurdles. To increase the emphasis of software development within a project, high quality software can be published in journals specialized in certain types of research software, like *Geoscientific Model Development* (https://www.geoscientific-model-development.net, last access: 13 December 2018) for numerical Earth system models or in more general journals for software like *The Journal of Open Research Software* (https://openresearchsoftware.

metajnl.com, last access: 13 December 2018) or *The Journal of Open Source Software* (http://joss.theoj.org, last access: 13 December 2018). This strengthens the recognition of the development work and contributes to a further reuse of the software, which saves resources for other projects.

Data availability. No data sets were used in this article.

Author contributions. Both authors made substantial contributions to this publication.

Competing interests. The authors declare that they have no conflict of interest.

Special issue statement. This article is part of the special issue "Project management in geosciences: systems and practices for high-impact research". It is a result of the EGU General Assembly 2018, Vienna, Austria, 8–13 April 2018.

Acknowledgements. We thank Robert Gieseke and anonymous reviewer for providing valuable comments. Nadine Wieters has received funding from the Initiative and Networking Fund of the Helmholtz Association through the project "Advanced Earth System Modelling Capacity (ESM)".

The article processing charges for this open-access publication were covered by a Research Centre of the Helmholtz Association.

Edited by: Sylvia Walter
Reviewed by: Robert Gieseke and one anonymous referee

References

Bowers, J. and Voors, M.: How to improve your relationship with your future self, Revista de Ciencia Política (Santiago), 36, 829–848, https://doi.org/10.4067/S0718-090X2016000300011, 2016.

Easterbrook, S. M. and Johns, T. C.: Engineering the Software for Understanding Climate Change, Comput. Sci. Eng., 11, 65–74, https://doi.org/10.1109/MCSE.2009.193, 2009.

Fitzpatrick, B., Pilato, C., and Collins-Sussman, B.: Version Control with Subversion, O'Reilly, 2004.

Fritzsch, B., Bernstein, E., Castell, W. Z., Diesmann, M., Haas, H., Hammitzsch, M., Konrad, U., Lähnemann, D., McHardy, A., Pampel, H., Scheliga, K., Schreiber, A., and Steglich, D.: Scientific Software – the role of best practices and recommendations, in: EGU General Assembly Conference Abstracts, vol. 19 of EGU General Assembly Conference Abstracts, 23–28 April 2017, Vienna, p. 3294, 2017.

Giorgetta, M. A., Brokopf, R., Crueger, T., Esch, M., Fiedler, S., Helmert, J., Hohenegger, C., Kornblueh, L., Köhler, M., Manzini, E., Mauritsen, T., Nam, C., Raddatz, T., Rast, S., Reinert, D., Sakradzija, M., Schmidt, H., Schneck, R., Schnur, R., Silvers, L., Wan, H., Zängl, G., and Stevens, B.: ICON-A, the Atmosphere Component of the ICON Earth System Model: I. Model Description, J. Adv. Model. Earth Syst., 10, 1613–1637, https://doi.org/10.1029/2017MS001242, 2018.

Goble, C.: Better Software, Better Research, IEEE Internet Comput., 18, 4–8, https://doi.org/10.1109/MIC.2014.88, 2014.

Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., and Wilson, G.: How do scientists develop and use scientific software?, in: 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, 23 May 2009, Vancouver, BC, Canada, 1–8, https://doi.org/10.1109/SECSE.2009.5069155, 2009.

Hettrick, S., Antonioletti, M., Carr, L., Chue Hong, N., Crouch, S., De Roure, D., Emsley, I., Goble, C., Hay, A., Inupakutika, D., Jackson, M., Nenadic, A., Parkinson, T., Parsons, M. I., Pawlik, A., Peru, G., Proeme, A., Robinson, J., and Sufi, S.: UK Research Software Survey 2014, Tech. rep., University of Edinburgh on behalf of Software Sustainability Institute, Edinburgh, https://doi.org/10.7488/ds/253, 2015.

Johanson, A. and Hasselbring, W.: Software Engineering for Computational Science: Past, Present, Future, Comput. Sci. Eng., 20, 90–109, https://doi.org/10.1109/MCSE.2018.021651343, 2018.

Katerbow, M. and Feulner, G.: Recommendations on the development, use and provision of Research Software, Zenodo, https://doi.org/10.5281/zenodo.1172988, 2018.

Rochkind, M. J.: The source code control system, IEEE T. Softw. Eng., SE-1, 364–370, https://doi.org/10.1109/TSE.1975.6312866, 1975.

Wang, Q., Danilov, S., Sidorenko, D., Timmermann, R., Wekerle, C., Wang, X., Jung, T., and Schröter, J.: The Finite Element Sea Ice-Ocean Model (FESOM) v.1.4: formulation of an ocean general circulation model, Geosci. Model Dev., 7, 663–693, https://doi.org/10.5194/gmd-7-663-2014, 2014.

Wilson, G.: Software Carpentry: lessons learned [version 2; referees: 3 approved], F1000Research, 3, https://doi.org/10.12688/f1000research.3-62.v2, 2016.