# The Software Engineering Initiative of DLR

Overcome the obstacles and develop sustainable software

Carina Haupt
German Aerospace Center (DLR)
Berlin, Germany
carina.haupt@dlr.de

Tobias Schlauch
German Aerospace Center (DLR)
Braunschweig, Germany
tobias.schlauch@dlr.de

Michael Meinel
German Aerospace Center (DLR)
Berlin, Germany
michael.meinel@dlr.de

## ABSTRACT

Software is a vital part of modern research. The competence to develop sustainable software becomes increasingly important for research organizations. The DLR - a large research organization in Germany - has set up a software engineering initiative to address typical obstacles in this regard such as missing long-term funding, lack of incentives, or missing knowledge about essential software development practices. In this paper, we describe the concept and activities of the initiative as well as discuss the impact of these activities on the identified obstacles.

## CCS CONCEPTS

• **Software and its engineering → Software creation and management**; **Maintaining software**; **Collaboration in software development**;

## KEYWORDS

Research Software Engineering, Obstacles, Maintenance, Community

**ACM Reference Format:**
Carina Haupt, Tobias Schlauch, and Michael Meinel. 2018. The Software Engineering Initiative of DLR: Overcome the obstacles and develop sustainable software. In *Proceedings of 2018 International Workshop on Software Engineering for Science (SE4Science'18).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

## 1 INTRODUCTION

Using and developing software increasingly became part of the daily work of most scientists [4, 6]. While research and the reproducibility of its results rely on software [1, 5, 7], the development of sustainable software[1] is still not recognized as an important goal in many research facilities. Software is often seen as a tool which should fulfill its task. Particularly, software attributes which contribute to its sustainability like re-usability, maintainability, and extendability are not prioritized and often not even considered.

[1]In this context, we consider sustainability as the capacity of a software to endure.

*SE4Science'18, June 2, 2018, Gothenburg, Sweden*
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

The situation now changes due to the growing complexity of research problems which shall be addressed by software and therefore require complex software projects. However, this change only happens slowly and often starts by the initiative of single individuals. For example, these individuals are scientists who are tired of redeveloping features and working with unmaintainable software. Sometimes even management initiates corresponding changes due to concerns with rising software development costs. Particularly, when we considered the obstacles which hinder the development of sustainable software at the German Aerospace Center (DLR), we often encountered lack of resources, motivation, or knowledge as root causes.

The *DLR software engineering initiative* has been initiated to address the identified obstacles and challenges. It mainly focuses on improving sustainability and quality of software products by providing guidelines, development infrastructure, training, opportunities for knowledge exchange and consulting. The remaining paper is structured as follows:

- First, we characterize software development at DLR to illustrate the context (Sect. 2).
- Then, we describe the obstacles faced at DLR while developing sustainable software (Sect. 3).
- Finally, we present the software engineering initiative and describe how its activities help to overcome the identified obstacles (Sect. 4).

## 2 SOFTWARE DEVELOPMENT AT DLR

DLR is a large research organization in Germany and conducts research in the domains aeronautics, space, energy, transportation, and security. Software development plays an increasing role in DLR's research activities. Over 25% of DLR‘s personnel costs are spent on software development.

Reflecting the various research domains, there is also a wide range of developed software. Typically, it includes areas such as simulation and modeling, flight control, signal and data processing, knowledge and data management, visualization, communication, and administration. The diversity is also reflected by the programming languages and frameworks in use, which include Python, R, Perl, C, C++, Fortran, IDL, Matlab, LabView, Ada, Java, .Net, and others.

From the maturity point of view, we can differentiate between *small, focused, and reliable research software*, *large long-term maintained scientific frameworks* forming the basis for further research, as well as *production critical software.* Most software developed at DLR belongs to the group of small research software and scientific frameworks.

Typical development team sizes range from one up to 20 persons.

Typically, there is a small current team size but there have been many more contributors in the past. The typical developer group at DLR consists of one scientist supported by some students. They usually have no specific computer science background.

Unfortunately, research software is still often developed using an ad-hoc, code-and-fix approach without documentation, source code version control, or issue tracking. This approach is useful for learning and experimenting. From our perspective, this approach often fails when aiming to develop sustainable software or to exactly reproduce results.

## 3 OBSTACLES WHEN DEVELOPING SUSTAINABLE SOFTWARE

In the following, we describe the observed obstacles at DLR. They origin from our personal experiences, discussions during trainings and knowledge exchange workshops, as well as internal consultancy work. While the encountered obstacles are well known in general, they have a particular magnitude in research organizations due to typically missing organizational structures to support sustainable software development.

### 3.1 Lack of Resources

While software engineering activities and best practices if applied continuously [2] provide benefits in the long run, they require the availability of adequate resources. Lacking these resources is a big obstacle when aiming to develop sustainable software.

**Project-based funding:** Internal and external projects can be considered as the main founding sources for DLR‘s research institutes. The obvious problem with this temporally limited funding is that it does not cover required resources for software maintenance beyond the projects allocated duration. In addition, it is hard for scientists to cover software maintenance in follow-up projects as they are focused on new research goals. Finally, missing follow-up projects in combination with the temporary contracts of many researchers can lead to the loss of essential knowledge carriers and can become a serious problem for the sustainability of research software.

**Hard accessible long-term funding:** In addition to project-based funding, research institutes have a certain amount of basic funding which could be used for software maintenance. However, management of the research institute has to be convinced of the benefits to obtain it for this purpose. Depending on the concrete case, this does not only require appreciation of the direct supervisors, but even higher management levels. At DLR, we regularly observe that scientists cannot overcome this obstacle by themselves.

**Missing infrastructure:** The availability of an infrastructure providing essential development tools (e.g., version control systems) is important for an efficient development and maintenance of software. In addition, these tools have a high potential of re-use for different software developments. However, the costs for administration and support of users must not be underestimated. Particularly, research institutes which are traditionally rather unrelated with computer science struggle to provide these tools for their researchers.

### 3.2 Lack of Motivation

Success is often tied to the motivation of the involved persons. The main goal of scientists is to do research, not software development. But without software engineering practices, software sustainability tends to fail. For development of sustainable software, everybody needs to understand this aspect and be motivated to take respective steps.

**Unmotivated scientist:** For most scientists, software is a tool that they need for their research. Software has to fulfill its task. Resources are rare and preferably used in the direct scope of the current research project. Long-term goals like reusability, maintainability, and extendability of the software therefore are seen as not necessary and too much overhead.

**Unmotivated management:** Management of the research institutes often still underestimates the complexity of software development and the long-time value of sustainable software. Therewith they do not provide resources for software engineering activities. Without these resources, software development cannot be performed properly and the software‘s sustainability is endangered. While this is not the case everywhere, we still observe it frequently at DLR and other research organizations.

**Missing incentives:** The success of scientists and research institutes is tightly coupled to the success of their research activities. This success is measured by indicators, which are primarily focused on publications. In this context, sustainability of involved software and reproducibility of results still play a subordinate role. Another example are research-focused project calls. While software development is an explicit part and receives funding, software engineering activities and the sustainability of resulting software are often not explicitly covered. Missing incentives and credits for sustainable and reproducible research software substantially reduce the motivation of scientists and research institutes to invest in this aspect.

### 3.3 Lack of Knowledge

Research software is mainly developed by scientists who are domain experts. Most of them have no specific education in software development. Usually they had programming courses at university or they self-taught some programming skills. Thus, their knowledge about software engineering and adjacent topics is quite limited.

**Missing knowledge:** An obvious obstacle is that scientists do not know how to develop sustainable software. While they may have programming skills to some extent, they lack training in software engineering methods. Sometimes they do not even know about the existence of such methods.

**Missing strategy:** Introducing the full stack of software engineering methods to a project is a lot of work and often not necessary. Selecting the right methods and tools at the right time is a challenge and requires training as well as experience. We observed that untrained but highly motivated scientists tend to set their goals too high. Thereby they and their colleagues get frustrated by the overhead and loose interest.

## 4 THE DLR SOFTWARE ENGINEERING INITIATIVE

In 2005, the *software engineering initiative* has been started to improve the sustainability and quality of research software at DLR. Core of the initiative is the software engineering network. It consists of representatives from the different DLR research institutes and forms DLR‘s central exchange forum about software engineering. The software engineering network drives the direction of the different activities while taking into account the demands and opinions of the involved research institutes. In the following, we describe the main activities of the initiative and explain how they contribute to address the identified obstacles (Sect. 3).

### 4.1 Software Engineering Directive and Guidelines

The software engineering directive is part of the DLR quality management policy and mandatory for DLR‘s research institutes. Main focus of the directive is to state the importance of the topic for DLR‘s research activities. It demands from the research institutes to address the topic accordingly and to identify developed software which is important for their research. Software development and connected practices are strongly influenced by organizational constraints and the domain of the developed software. Therefore, it is important that research institutes develop the topic in their context. In essence, it demands activities like community building, investing in required development infrastructure as well as investments in improving software development skills of scientists. The software engineering network representative of the institute is in charge of driving these aspects.

As the directive is part of DLR‘s quality policy, its implementation is checked by yearly quality audits. Thus, the directive serves as a constant reminder for the institute management with regard to sustainable software development. Focus of these audits is the development of the topic at the institute. In the long run, we hope that the directive helps to stepwise start organizational changes that help to improve the conditions for research software sustainability.

The software engineering guidelines [3] have been published as part of the directive and describe DLR‘s understanding of good software development practices. Their primary focus is on supporting scientists in software development. The guidelines give advice in different topics like requirements management, change management, implementation, and software test. Every topic is introduced and defines a set of recommendations. In addition, recommendations are motivated and give hints about their implementation. To simplify the usage, we provided checklists and developed a simple classification scheme consisting of three maturity levels. The scheme takes aspects into account like criticality, software scope, software distribution, and period of development. It helps to filter the recommendations and to fit them into the right context.

The main goal of the guidelines is to provide an entry point for scientists into software development and engineering. Besides providing knowledge, the guidelines help scientists to judge which recommendations are required at a certain point of time. In addition, they help them to justify required resources. Finally, the guidelines support the research institutes to develop the overall topic accordingly. Particularly, it helps them to identify required skills and infrastructure when developing software classified in a certain maturity level.

### 4.2 Development Infrastructure

It has been early recognized that the availability of basic development tools is essential to motivate scientists to follow software development practices. However, professional operation including proper user support cannot be efficiently performed by a single research institute. Thus, in cooperation with DLR‘s central IT management, the version control system Subversion[2] and the issue tracker MantisBT[3] have been provided free of charge at DLR since 2005. Particularly, Subversion has been well received with currently about 1.300 active repositories.

Currently, we are in progress to modernize the development infrastructure by introducing the software forge GitLab[4]. In this context, we focus on support for collaboration and automatic testing. Besides a much more accessible solution, the collaboration functionalities form the basis to further connect scientists.

### 4.3 Trainings

Trainings support direct and interactive knowledge transfer. In this context, a regular training is offered as part of DLR‘s education program. It focuses on structured software development and practical application of typically used development tools. The participants can practice all aspects in an example project. Topics and tools are aligned with the software engineering guidelines (Sect. 4.1) and the DLR development tools (Sect. 4.2). The training is regularly updated to fit the current state of the art and DLR requirements.

Target group of the training are scientists working alone or in small development teams. Main goal is to provide knowledge and hands-on experience about development of sustainable software. In addition, we want to enable the participants to transfer the new knowledge directly into a suitable strategy for their projects. Therefore, two trainers instruct ten to fifteen participants in this two-day training to provide an intensive individual support.

The trainings are generally well visited and received by the participants [3]. Beyond the provision of knowledge, the training helps participants to get in contact with other scientists that develop software for their research.

### 4.4 Opportunities for Knowledge Exchange

Research organization such as DLR employ a high number of university graduates with limited work experience and varying software development skills. Opportunities for knowledge exchange with other practitioners help them to get properly started with their work. In addition, those opportunities help to connect senior scientists working in similar domains but different research institutes.

In this context, the SoftwareEngineering.Wiki is the single point of access for software engineering related information at DLR. It provides information about software engineering related topics, tools, best practices, literature, and events. In addition, a moderated questions-and-answers section is provided which often forms the basis for new wiki content. The SoftwareEngineering.Wiki is an

[2]https://subversion.apache.org/
[3]https://www.mantisbt.org/
[4]https://about.gitlab.com/

actively used and steadily growing information resource [3]. Easy information access, the existence of a community around the wiki, and the option for help via the questions-and-answers section lower the barrier to get in contact with others and to contribute content. Reading about positive experiences and examples helps to increase motivation to introduce new software engineering methods.

In addition, a knowledge exchange workshop series [3] with focus on software engineering has been started since 2014. In total about 50 scientists participate in each workshop. Each year the workshop focuses on a different topic like development processes, open source, software architecture, or embedded systems. The workshop is aiming to provide knowledge, experience exchange and networking. Results are captured in the SoftwareEngineering.Wiki to preserve them and to allow those who could not attend to benefit from them. Particularly, we aim to involve and connect the participants. Therefore, participants can contribute experience reports, technical presentations, or lightning talks. In addition, we always include an interactive session and a social event to support networking activities. The workshops are generally positively received by the participants. Specifically, they are considered as a big source of motivation and helpful to advance the knowledge. The official character of the workshop series also increases management‘s perception of software engineering as important for research.

## 4.5 Consulting

Consulting is focused on scientists with limited background in software development. In this context, we provide a pool of experienced software engineers that participate in concrete projects. Their primary role is usually to set up the development environment, to propose a software development process, and to support scientists in creating high quality software. The software engineering guidelines (Sect. 4.1) are used as basis to discuss and plan the details of these activities.

As a specific variant of consulting, an experienced software engineer works on-site in a different research institute to help addressing the requirements of the software engineering directive. There, he analyzes existing software projects and processes. Based on these observations a stepwise plan is developed to improve the quality of software engineering. It often results in activities like performance of individual trainings, introduction of new tools, or support in specific questions.

Because of the consultancy work, scientists gain a good insight into the benefits of software engineering practices which in turn improves their motivation to follow them. In addition, we observe a growing number of research projects that involve software engineering experts from the very beginning. Finally, consulted institutes tend to employ more software engineering specialists to follow up the achieved results.

## 5 CONCLUSIONS

Raising the awareness and supporting the establishment of organizational structures for sustainable software development are the main focus of the DLR software engineering initiative. Further supporting activities geared to scientists complement theses aspects and are still evolving since the start of the initiative. Finally, we want to summarize how these activities currently address the obstacles identified in Sect. 3.

In context of the obstacle *lack of resources*, there is still room for improvement. Particularly, a traditional boundary condition like project-based funding cannot simply be changed and also depends on external factors. However, the software engineering directive and guidelines already show improvements. They raise the awareness of the topic at the level of the research institute management and help to make clear the minimum of required software development activities in a specific case. These aspects increase the number of internal software-related projects that explicitly include software engineering activities in their planning. For example, the internal technology marketing department started to explicitly require software engineering work packages if a software-related project wants to receive its funding. In addition, the official character of the guidelines helps scientists to better express the relevance for long-term funding of software engineering activities. Finally, the central availability of professionally supported, commonly required development infrastructure has been an important basis to get scientists started with base practices.

In context of the obstacle *lack of motivation*, we observed that activities focused on collaboration and support lead to improvements. For example, the exchange workshop, the trainings, and the wiki help to connect scientists in context of software development. Finding others with similar problems and discussing about the approaches taken help to identify benefits and motivate to try them. In similar way, direct discussion with experts in context of consulting and training can increase motivation. Finally, the software engineering directive is an incentive for the institute management to address the topic. In addition, its official character encourages scientists to claim support.

The software engineering guidelines form the basis to address the obstacle *lack of knowledge* by providing recommendations and supporting the scoping. In addition, activities like training, consulting, performance of exchange workshops, and the wiki help scientists to practically apply the knowledge.

## REFERENCES

[1] Christian Collberg, Todd Proebsting, and Alex M Warren. 2015. *Repeatability and Benefaction in Computer Systems Research - A Study and a Modest Proposal.* techreport. University of Arizona.

[2] Brian Fitzgerald and Klaas-Jan Stol. 2014. Continuous Software Engineering and Beyond: Trends and Challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE 2014).* ACM, New York, NY, USA, 1–9. https://doi.org/10.1145/2593812.2593813

[3] Carina Haupt and Tobias Schlauch. 2017. The Software Engineering Community at DLR: How we got where we are, In Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE5.1), Neil Chue Hong, Stephan Druskat, Robert Haines, Caroline Jay, Daniel S. Katz, and Shoaib Sufi (Eds.). *Proceedings of the Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE5.1).* http://elib.dlr.de/114050/

[4] James Howison and Julia Bullard. 2016. Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. *Journal of the Association for Information Science and Technology* 67, 9 (2016), 2137–2155. https://doi.org/10.1002/asi.23538

[5] Jelena Kovacevic. 2007. How to encourage and publish reproducible research. *IEEE International Conference on Acoustic, Speech Signal Processing* 4 (April 2007), 1273 –1276.

[6] Udit Nangia and Daniel S. Katz. 2017. Track 1 Paper: Surveying the U.S. National Postdoctoral Association Regarding Software Use and Training in Research. (Aug 2017). https://doi.org/10.6084/m9.figshare.5328442.v3

[7] Patrick Vandewalle, Jelena Kovacevic, and Martin Vetterli. 2009. Reproducible research in signal processing. *IEEE Signal Processing Magazine* 26, 3 (May 2009), 37 – 47.