Track 1 Paper: The Software Engineering Community at DLR — How we got where we are

Carina Haupt & Tobias Schlauch
German Aerospace Center (DLR)
Berlin & Braunschweig, Germany
carina.haupt@dlr.de, tobias.schlauch@dlr.de

***Abstract*—Sustainable software and reproducible results become vital in research. Awareness for the topic as well as the required knowledge cannot be taken for granted. We show, how scientists at DLR are encouraged to form a self-reliant software engineering community and how we supported this by providing information resources and opportunities for collaboration and exchange.**

I. Introduction

Research software became one of the basic tools of science. An increasing amount of scientists needs to develop software to support their research. Thus, many researches are faced with topics outside their domain: The development of sustainable software which delivers reproducible results.

The German Aerospace Center (DLR) conducts research and development in the domains aeronautics, space, energy, transportation, and security. Software development plays an increasing role in DLR's research activities. Around 2000 to 3000 persons of DLR's 8000 employees develop software – in part or full time. Important challenges thereby are to support development teams from different research domains and to establish basic quality of produced software.

Particularly, the lack of knowledge about software development leads to many issues such as missing source code, unclear source code, missing documentation, or substantial bugs. Scientists are often not aware of the challenges of software development or its implications. Thus, they are at risk to not achieve repeatability and reproducibility of their scientific results based on the developed software [Kov07], [VKV09], [CPW15].

To make this knowledge accessible and to support DLR scientists in applying it, we initiated the *DLR software engineering initiative* to improve sustainability and quality of software products. We started activities and provided information resources, which we describe in the remaining paper as follows:

- We characterize software development at DLR to illustrate the context (Sect. II).
- We describe provided information resources about software engineering and the training activities to make that knowledge available (Sect. III).
- We explain our efforts for knowledge sharing among peer scientists by providing collaboration tools and organizing dedicated knowledge exchange workshops (Sect. IV).

II. Software Development at DLR

There is a wide range of software developed at DLR, which 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 in use, which include Python, R, Perl, C, C++, Fortran, IDL, Matlab, LabView, Ada, Java, .net, and others.

Software is often developed using the *code-and-fix* approach [Boe88] without documentation, source code version control, or issue tracking. This approach can be successful for throw-away prototypes. It is particularly useful for learning and experimenting. From our perspective, this approach often fails when the development progresses. In addition, it makes it difficult to ensure repeatable and reproducible results.

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 projects in DLR belong to the group of small research software and scientific frameworks.

Typical development team sizes range from one up to 20 persons. However, at DLR we often have one person teams. Usually, there is a small current team size but there have been many more contributors in the past. Researchers supported by students form the typical developer group at DLR. However, they usually have no specific computer science background.

The aim of the software engineering initiative is to overall improve DLR's competence in the area of software development and to build a self-reliant community.

III. Information Resources and Education

A major source of problems has been the lack of knowledge about software engineering. To overcome this and to enhance knowledge sharing within DLR, we provided three information resources as a foundation for all software development activities: a set of software engineering guidelines to offer scientists suitable starting points, a Wiki space providing information about software engineering related topics, and regular training courses about development processes and tools.

A. Software Engineering Guidelines

In 2016, the software engineering guidelines to support scientists have been published. These guidelines give advice

in different fields of software development like change management, design and implementation, and software test. Every topic is introduced and necessary software engineering terms are explained. For every recommendation the motivation and reasoning is described. Basically, the guidelines focus on good development practices and help to retain essential development knowledge. The latter aspect is particularly important within research organizations because of the typical employee fluctuation. Overall, there are currently 78 recommendations which are regularly reviewed and revised. These recommendations describe DLR's current understanding of good software development practices.

To make it easy to start with the guidelines, we developed a simple classification scheme consisting of three maturity levels. The scheme takes aspects into account like software size, expected software lifetime as well as criticality. It is used to filter the recommendations to a necessary minimum and to fit them into the right context. In addition, we created checklists to offer a light-weight tool to use the guidelines in practice. The checklists are available in different formats (e.g., Markdown, reStructuredText, Confluence Wiki, Word) to fit the individual working environment. For example, the Markdown version can be easily managed in version control systems alongside with the software.

In the following, we describe the application of the guidelines at the example of a stand-alone Python script for data conversion which shall be re-used and further developed. According to the classification scheme, the script fits into the first maturity level. The aim of this level is to allow a new developer to effectively take over development. In this case, the basic list of recommendations is reduced to 23. On this basis, further relevant recommendations may be added or irrelevant recommendations may be removed. This tailoring step is essential to map the recommendations to the concrete development context. The author of the software usually performs this task. In addition, there are support contacts established in the DLR institutes. Without any adaptation the recommendations of the first maturity level can be summarized as follows:

- Manage the code using a version control system, properly work with the corresponding repository, and make sure that ideally everything is contained to build and test the software.
- Provide a build script or similar to automatically create an executable version.
- Consistently apply a basic coding style, strive for a modular design, and avoid code duplication.
- If you release the software internally, you should make sure that the software works as expected and indicate the release with a proper release number.
- If you release the software outside DLR, you have to perform an intellectual property check (e.g., licenses of used software dependencies).
- Document the following aspects:
  - Short description of the software purpose.
  - Basic steps and requirements to build and use the software.
  - Basic steps and requirements to contribute to development.
  - Central concepts and constraints not obvious from code.
  - Known problems and improvement ideas.

In case of the Python script, an implementation of these recommendations can be as follows:

- The code is managed in a Git repository.
- The Python script works standalone. However, a build script is provided to ease packaging and usage.
- The source code is formatted in accordance to PEP 8[1] and breaks down the performed tasks into separate functions.
- The author performed basic tests and added examples about valid inputs and expected results to the repository. In addition, the current version is indicated with a release number according to the semantic versioning scheme[2].
- The script is only used internally. Therefore an intellectual property check is not required yet. However, in general, we encourage it at an early stage.
- A *README.md* file describes the purpose of the script and basic usage information. In addition, the required Python version and the supported operating system are stated. Finally, changes and improvements of different releases are explained as well.

This example shows that the guidelines encourage good development practices but are still flexible enough to fit the concrete development context. However, this tailoring step requires experience and guidance. Therefore, other activities described in this paper are important for proper practical adoption of the guidelines at DLR.

B. *SoftwareEngineering.Wiki*

In 2013, we started the SOFTWAREENGINEERING.WIKI space. It provides information about software engineering related topics, tools, best practices, literature, questions (Sect. IV-A), and events. Questions and their answers are often used as basis to start new topics or tool pages. Since asking a question is a much lower hurdle to take than starting a Wiki page, this section became the entry point for a lot of contributors. The Wiki also offers a blog. It is used to announce new Wiki content and to share news, such as relevant call for papers or workshops.

The SOFTWAREENGINEERING.WIKI is the focal point for the DLR software engineering community. It allows us to provide a single point of access for software engineering related information at DLR. In addition, it enables everybody to share knowledge and experiences. Currently the SOFTWAREENGINEERING.WIKI consists of 315 pages. A number which grows steadily (Fig. 1), even with a currently decreasing growth rate due to the switch from mainly page creation to page changes. The majority of the content is provided by

[1]Python style guide: https://www.python.org/dev/peps/pep-0008/
[2]Introduction to semantic versioning: http://semver.org/

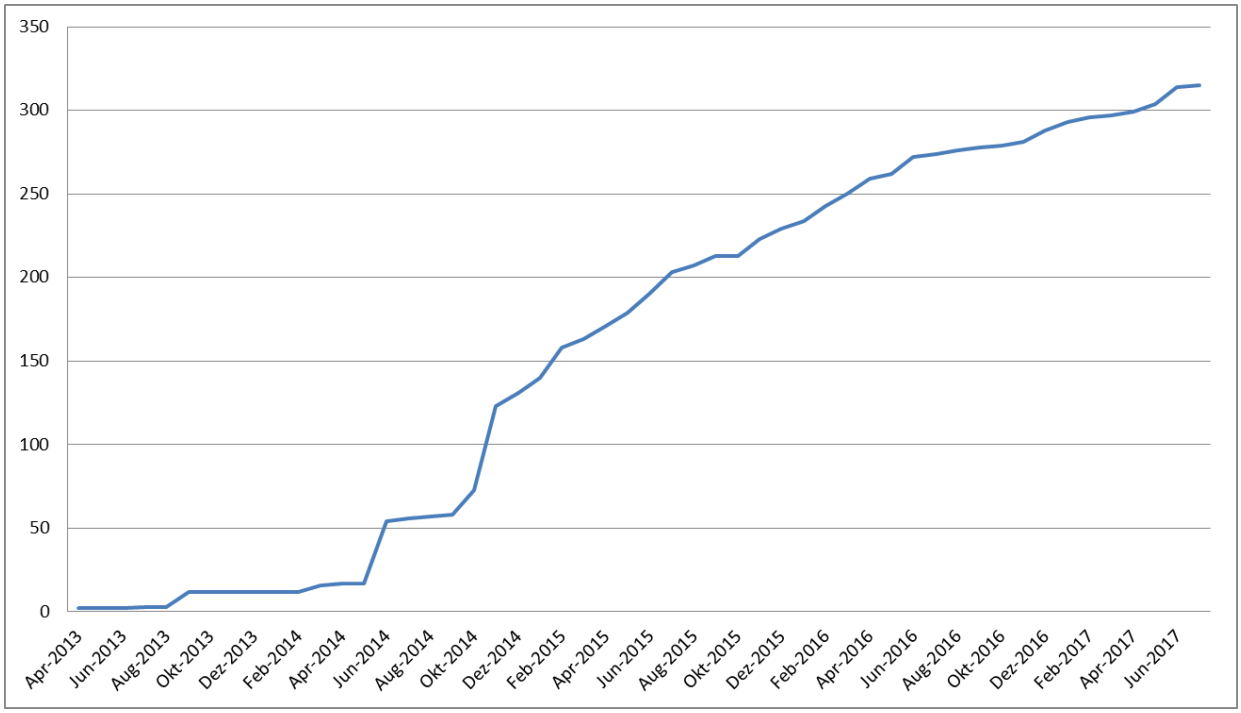

Figure 1: Number of pages in the SOFTWAREENGINEERING.WIKI

people working at DLR. Maintenance of the Wiki structure and moderation is done by a small group.

C. Trainings

Regular trainings have been offered since 2009. Since 2013, they are part of the DLR education program. The trainings take place at different DLR locations across Germany. These distributed trainings allow scientists to participate in a training close to their location and reduce travel overhead. In addition, several customized trainings have been held for specific departments or teams.

The regular training (*"Tool supported Software Development"*) focuses on structured software development and practical application of typically used development tools (e.g., version control system, issue tracker, build tool). The participants can directly practice all aspects in a hands-on example project. Topics and tools are aligned with the DLR landscape and are regularly updated to fit the current state of the art and DLR requirements. Two trainers instruct ten to fifteen participants in this two-day training.

The ratings of the trainings are either good or very good (Fig. 2). In total, the overall rating and the rating of matched expectations is 1.5. Ratings for transferability into daily work and relevance for daily work are lower but still considered good, with 1.97 and 2.16. One reason for the lower ratings may be the focus on software development teams while scientists often develop on their own. We decided to keep the focus on small teams as chances are good that at least students join periodically. In addition, most practices can be easily adapted to one person teams.

The human resources department supports performance of the regular training as every other educational offering at DLR. For example, it provides required infrastructure such as rooms and services such as advertisement and participant management. In this context, scientists decide whether they require such training or not. Thus, it is important to convince them and to further raise the awareness of this topic. In this regard, we plan to further improve the regular trainings by creation of a graded curriculum aligned with the software engineering guidelines, clearly indicating typically required skills for specific maturity levels, and improving internal advertisement activities.

IV. KNOWLEDGE SHARING

Besides providing information and trainings, an environment for *sharing knowledge between peer scientists* is most important for the DLR's software engineering community building strategy. Next to the SOFTWAREENGINEERING.WIKI (Sect. III-B), which offers collaboration possibilities due to its nature, we introduced two additional elements to foster knowledge sharing. A Web-based support tool using the Wiki and a workshop series for face-to-face knowledge exchange and discussions.

A. Community Support

In November 2014, an internal "Ask a question" section has been created in the SOFTWAREENGINEERING.WIKI (Sect. III-B) to answer software engineering related questions. The advantage of this Wiki section is that it enables everybody at DLR to provide answers and advice, to share experiences,

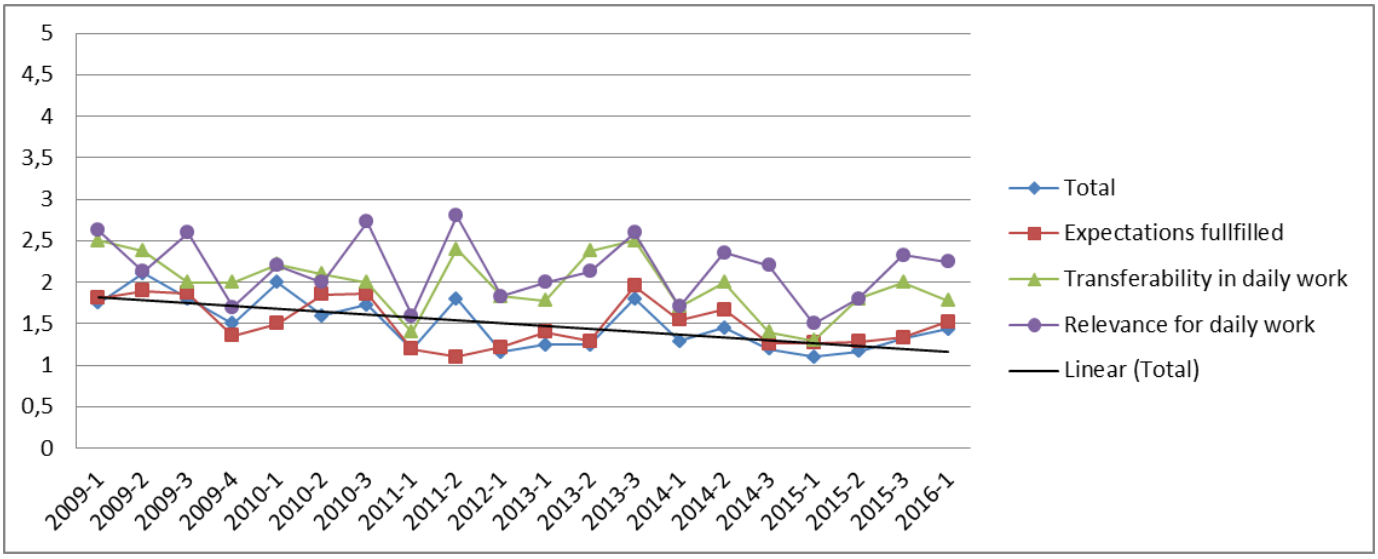

Figure 2: Extract of the feedback for all *"Tool supported software development"* trainings given since 2009. (1=very good, 5=very bad)

and to discuss. Additionally, everybody can benefit from the given answers.

22 questions have been asked and answered since November 2014. In average, there are 3.5 replies indicating that discussions have been started. Those discussions often resulted in creation of new pages. Questions generally belong to one of two types: Complex or DLR-specific. Particularly for the latter type, answers cannot be found by using external services such as StackOverflow or Google. Thus, the "Ask a question" section fills an important niche which cannot be filled by external services.

B. Knowledge Exchange Workshops

Since 2013, *knowledge exchange workshops* have been established at DLR. These workshop series are interdisciplinary and focus on long-term exchange of knowledge on overlapping topics. The goal is to establish and expand professional networks at DLR. Topics are proposed by DLR employees. Every employee can vote for a proposal. A selection committee — including the chair of the DLR Executive Board — makes the final decision based on the votes.

The knowledge exchange workshop series on software engineering started in 2014. Although the topic has not been selected by the committee, the proposing employees organized it anyway. The series is still ongoing thanks to a growing number of organizers. Each year the workshop focuses on a different topic like development processes, testing, open source, embedded systems, or architecture. Focus of the workshop is to provide knowledge, experience exchange and networking. Therefore it consists of experience reports, technical presentations, lightning talks, group work, and keynotes. Such keynotes generally are held by invited external speakers. In total about 50 scientists participate in each workshop. Results are captured in the SOFTWAREENGINEERING.WIKI (Sect. III-B) to keep them and to allow those who could not attend to benefit from them.

V. CONCLUSION

We presented our strategy and efforts to create a software engineering community for domain scientist at DLR. Focus of our initiative is to provide suitable opportunities for collaboration and exchange as well as a foundation of knowledge to work on. We started with the introduction of basic software engineering tools, accompanied with training courses in the regular education program. After that, we introduced collaboration platforms for knowledge sharing among peer scientists, such as the Wiki space and knowledge exchange workshops.

Combined, these information resources and activities enable the scientists to build a software engineering community, not relying on a single specialized department, but on each other. Since we started our initiative, an active community could be established and the awareness for the need of sustainable software delivering reproducible results grew.

Feedback of domain scientist is good or very good. Based on that feedback we extend our efforts in the future, for example, by providing more technical infrastructure for collaborative software development and to focus on the topic "inner source" to further strengthen the community.

REFERENCES

[Boe88] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61 – 72, May 1988.

[CPW15] Christian Collberg, Todd Proebsting, and Alex M Warren. Repeatability and benefaction in computer systems research - a study and a modest proposal. techreport, University of Arizona, February 2015.

[Kov07] Jelena Kovacevic. How to encourage and publish reproducible research. *IEEE International Conference on Acoustic, Speech Signal Processing*, 4:1273 –1276, April 2007.

[VKV09] Patrick Vandewalle, Jelena Kovacevic, and Martin Vetterli. Reproducible research in signal processing. *IEEE Signal Processing Magazine*, 26(3):37 – 47, May 2009.