Software Engineering Guidelines for Scientists

A Practical Handout for the Developing Researcher

Carina Haupt (@caha42)
German Aerospace Center (DLR), Simulation and Software Technology

2019 SIAM Conference on Computational Science and Engineering
Scientific Software: Practices, Concerns, and Solution Strategies

DLR

Software development at the German Aerospace Center (DLR)

Numbers

- More than 8000 employees
- ~20% of DLR employees involved in software development

Some Characteristics

- Variety of
  - Fields
  - Maturity
  - Software technologies
  - Team sizes
  - Backgrounds

We started a Software Engineering Initiative for DLR because we believe that our research results profit from better software!

SEI @ DLR

Software Engineering Initiative for DLR

- Networking and Collaboration
- Guidelines and Tools
- Trainings and Consulting
- Knowledge and Experience Exchange

The DLR Software Engineering Guidelines
Topics

Guidelines support **research software developers to self-assess their software** concerning **good development practices**.

- Guideline document & Checklists
- Joint development with focus on **good practices**, **tools**, and **essential documentation**
- **77 recommendations** give advice in different fields of software engineering:

Requirements Management

Software Architecture

Design & Implementation

Change Management

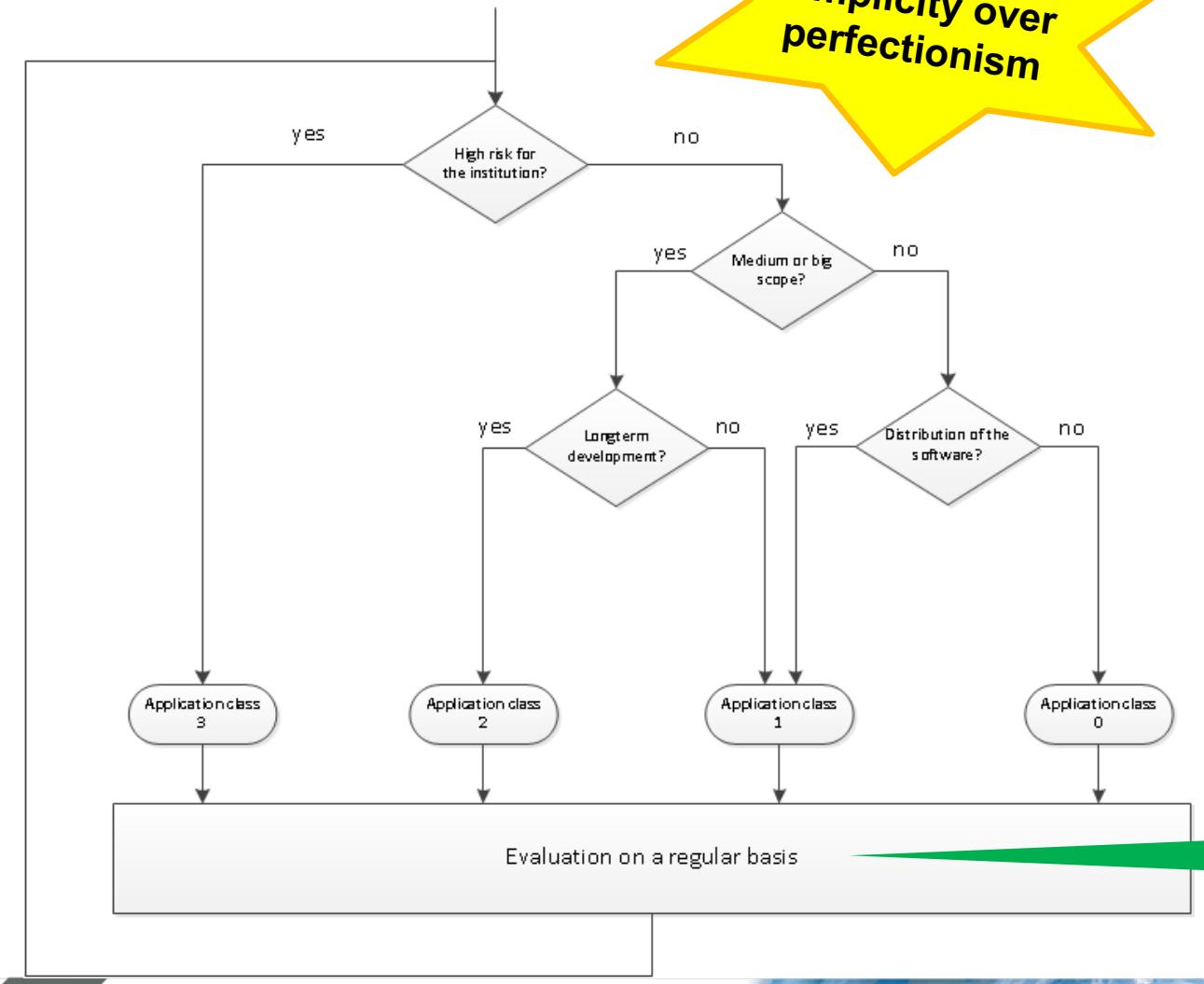
Software Testing

Release Management

Automation & Dependencies

https://zenodo.org/record/1344612 (EN)
https://zenodo.org/record/1344608 (DE)

The DLR Software Engineering Guidelines
Tailoring Checklists



An application class provides an initial starting point. Recommendations can be added and removed to fit the context.

Application class 1
- „small“, but other use it

Application class 2
- „medium – large“, other use it, long-term support

Application class 3
- „products“, critical for success of department or institute

Application class 0
- Personal „use“ (intentionally left blank)

Classification may change over time!

The DLR Software Engineering Guidelines

Using Checklists

Check List

Änderungsmanagement		
EÄM.2: Die wichtigsten Informationen, um zur Entwicklung beitragen zu können, sind an einer zentralen Stelle abgelegt.		
EÄM.5: Bekannte Fehler, wichtige ausstehende Aufgaben und Ideen sind zumindest stichpunktartig in einer Liste festgehalten und zentral abgelegt.		
EÄM.7: Ein Repository ist in einem Versionskontrollsystem eingerichtet. Das Repository ist angemessen strukturiert und enthält möglichst alle Artefakte, die zum Erstellen einer nutzbaren Version der Software und deren Test erforderlich sind.		
EÄM.8: Jede Änderung des Repository dient möglichst einem spezifischen Zweck, enthält eine verständliche Beschreibung und hinterlässt die Software möglichst in einem konsistenten, funktionierenden Zustand.		

Concrete Guideline

EÄM.7 Ein Repository ist in einem Versionskontrollsystem eingerichtet. Das Repository ist angemessen strukturiert und enthält möglichst alle Artefakte, die zum Erstellen einer nutzbaren Version der Software und deren Test erforderlich sind.	**ab 1**	Das Repository ist der zentrale Einstiegspunkt in die Entwicklung. Dadurch sind alle wesentlichen Artefakte sicher gespeichert und an einer Stelle auffindbar. Einzelne Änderungen können nachvollzogen und dem jeweiligen Urheber zugeordnet werden. Darüber hinaus stellt das Versionskontrollsystem die Konsistenz aller Änderungen sicher. Die Verzeichnisstruktur des Repository sollte man anhand bestehender Konventionen ausrichten. Quellen dafür sind typischerweise das Versionskontrollsystem, das Build-Werkzeug (vgl. Abschnitt 4.8 Automatisierung und Abhängigkeitsmanagement) oder die Community der eingesetzten Programmiersprache bzw. des verwendeten Frameworks. Dazu zwei

Topic Overview

4.4 Änderungsmanagement

Gegenstand des Änderungsmanagements[11] ist, systematisch und nachvollziehbar Änderungen an der Software durchzuführen. Ursachen für Änderungen sind beispielsweise Anforderungen, Fehler oder Optimierungen. Das Änderungsmanagement unterstützt dabei, den Überblick über den Entwicklungsstand zu behalten und die verschiedenen Entwicklungsaufgaben zu koordinieren.

In diesem Zusammenhang beschreibt der **Änderungsprozess**, wie **Änderungswünsche** (z.B. Anforderungen, Fehler, Optimierungen) prinzipiell auf Entwicklerseite abgearbeitet werden und anschließend ggf. in Form einer neuen Software-Version zur Verfügung stehen. Dieser Prozess ist im Detail in jedem Entwicklungskontext unterschiedlich. Daher ist es wichtig, diesen im Entwicklungsteam abzustimmen und kontinuierlich zu verbessern. In der Praxis ist darauf zu achten, dass sich die Abläufe effizient umsetzen lassen. Daher ist auf angemessenen Einsatz von Werkzeugen und Automatisierung zu achten.

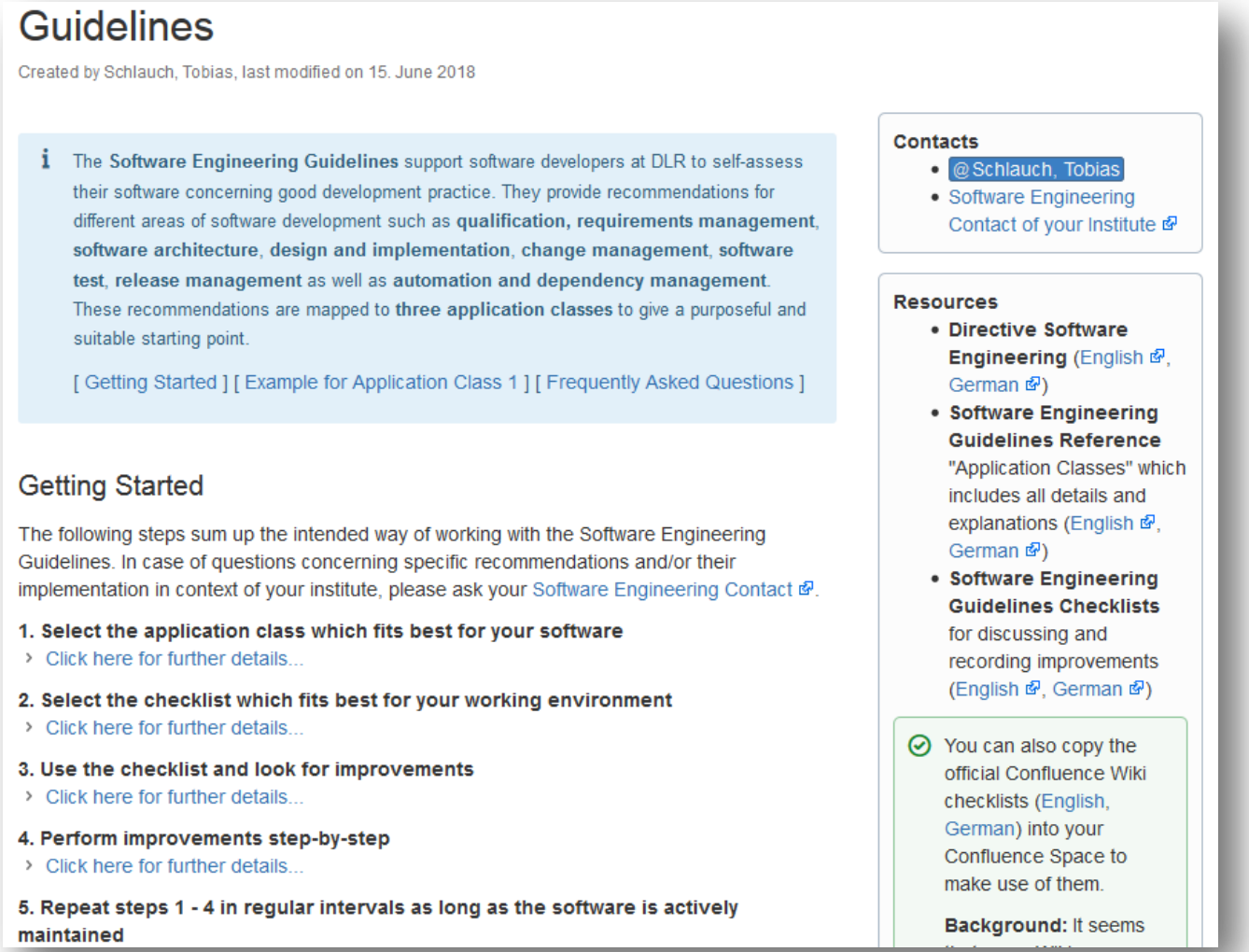
How do we use the guidelines?

Main use cases:

- Find out about the current status
- Identify improvements

Example situations:

- Find initial start point in new or legacy software projects
- Ongoing improvement
- Supporting hand-over
- Convince others
- Indicate applied practices to a third-party



Guidelines

Created by Schlauch, Tobias, last modified on 15. June 2018

The **Software Engineering Guidelines** support software developers at DLR to self-assess their software concerning good development practice. They provide recommendations for different areas of software development such as **qualification, requirements management**, **software architecture**, **design and implementation**, **change management**, **software test**, **release management** as well as **automation and dependency management**. These recommendations are mapped to **three application classes** to give a purposeful and suitable starting point.

[Getting Started] [Example for Application Class 1] [Frequently Asked Questions]

Getting Started

The following steps sum up the intended way of working with the Software Engineering Guidelines. In case of questions concerning specific recommendations and/or their implementation in context of your institute, please ask your Software Engineering Contact.

1. **Select the application class which fits best for your software**
   › Click here for further details...
2. **Select the checklist which fits best for your working environment**
   › Click here for further details...
3. **Use the checklist and look for improvements**
   › Click here for further details...
4. **Perform improvements step-by-step**
   › Click here for further details...
5. **Repeat steps 1 - 4 in regular intervals as long as the software is actively maintained**

Contacts
- @Schlauch, Tobias
- Software Engineering Contact of your Institute

Resources
- **Directive Software Engineering** (English, German)
- **Software Engineering Guidelines Reference** "Application Classes" which includes all details and explanations (English, German)
- **Software Engineering Guidelines Checklists** for discussing and recording improvements (English, German)

You can also copy the official Confluence Wiki checklists (English, German) into your Confluence Space to make use of them.

Background: It seems

Two recent examples…

Example 1: Support researchers improving a legacy Matlab code

Context:

- Matlab toolbox for image processing and analysis
- Legacy code base, one researcher + students ► close to class 1

Involvement of our RSE group:

- Make it “production-ready”, team development ► class 2
- Consulting: Set up processes and tools, training, no feature development
- Challenges: (legacy) Matlab, RSEs and developers at different sites

Example 1: Support researchers improving a legacy Matlab code (cont.)

Approach:

- Moving to GitLab
- Iterative process refinement
- Improving documentation and testing

Experiences:

- Checklists worked pretty well ► focus, discussions, status
- Remote consulting ► hard to assess “real” status
- Legacy code ► harder to make the right judgement

Example 2: New development of a metrics calculation tool for satellite performance

Context:

- Originally: New development of a metrics tool in Python 3
- But: Python 3 legacy code re-use required

Involvement of our RSE group:

- Develop a “production-ready” tool ► class 2/3
- Development and consulting: Feature development, set up processes and tools, supporting individual developers
- Challenges: (legacy) Python, scattered development team, many partners, hard time constraints

Example 2: New development of a metrics calculation tool for satellite performance (cont.)

Approach:

- Involve all partners
- Establish professional development process and environment early
- Iterative process refinement
- Refactor legacy Python code as needed

Experiences:

- Checklists worked pretty well ► status, discussion
- Unforeseen effort ► “hidden” dependencies, environment
- Missing priority hints ► harder to set focus

Lessons learnt / Next steps

Guidelines help to find out about the status, to discover improvements as well as to focus activities and discussions but …

… are not the solely solution to increase software quality.
… are no beginners tool and some details require improvements:

- Better indicate priorities
- Make dependencies more transparent
- Direct links to (more) practical examples

Supportive environment is key!

- Community, team culture, mentors
- Tools and trainings

Do you want to find out more?

- RSE17 talk "Helping a friend out – Guidelines for better software"

- We published the reference guides:
  - German version: https://doi.org/10.5281/zenodo.1344608
  - English version: https://doi.org/10.5281/zenodo.1344612
  - https://rse.dlr.de

Source: Zenodo, https://zenodo.org/record/1344612