APPLICATION CLASSES IN THE DLR SOFTWARE ENGINEERING GUIDELINES

Stephan Druskat
German Aerospace Center (DLR), Institute for Software Technology

DOI 10.5281/zenodo.7620412

License CC-BY-4.0 International

DLR

Why does DLR need Software Engineering Guidelines?

Numbers

- Over 10,000 employees
- ~20% of employees develop software

Developer characteristics

- Often no training in software development
- Large amount of software projects
- Variety of technologies in use

How to support scientists in developing **sustainable** research software?

What do (DLR) RSEs need for their project?

- Version control (git)
- Collaborative development platforms (GitLab, GitHub, …)
- Open source licensing (Apache 2.0, MIT, (L)GPL, …)
- Software architecture
- Requirements engineering (?)
- Build systems (CMake, Autotools, Maven, …)
- Meta build systems (Spack, EasyBuild, Conda)
- Test frameworks (PyTest, xUnit, …)
- Continuous integration testing (GitHub Actions, gitlab-ci, …)
- Integrated development environments (IDEs, e.g. Eclipse, VSCode, PyCharm, …)
- etc.

The DLR Software Engineering Guidelines
rse.dlr.de

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

Checklist-based evaluation

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.

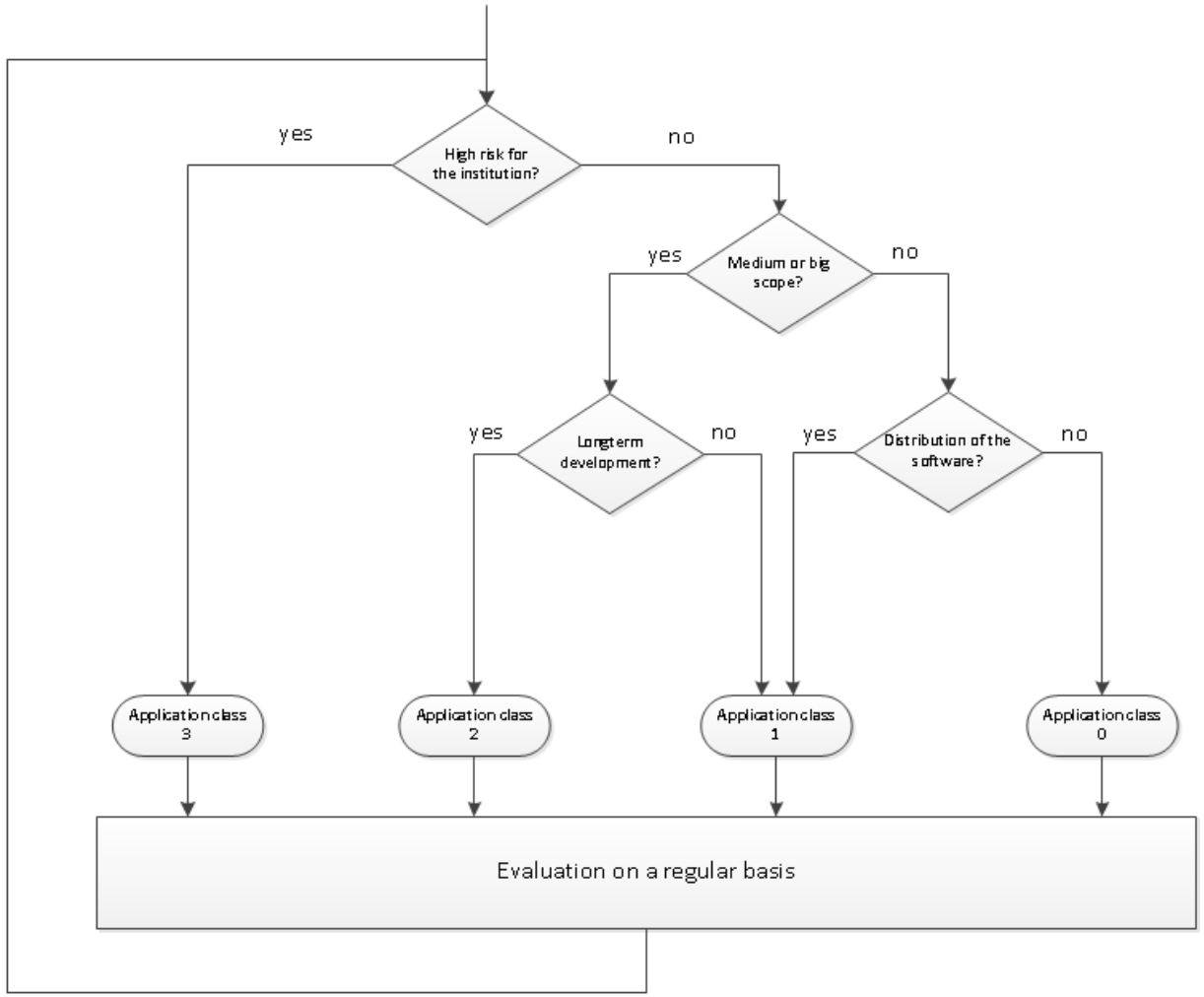
Application classes

Application classes provide an initial starting point.

Recommendations can be added and removed to fit the context.

- **Application class 0**
  - Personal "use" (intentionally left blank)
- **Application class 1**
  - "small", others (may) use it
- **Application class 2**
  - "medium – large", others use it, long(er)-term support
- **Application class 3**
  - "products", critical for success of department or institute

Application classes: classification process



Classification may change over time

Application classes: classification criteria

Criteria are not absolute

Application Class 1

- 1 Developer
- > 1 User
- Short-time relevance (i.e. until paper published)
- Group-internal usage
- Low risk
- Example: Paper accompanying data analysis script

Application Class 2

- > 1 Developer
- > 1 User
- Medium-time relevance (about 2-3 years)
- Institute-internal usage
- Medium risk
- Example: Internal standard data extraction toolkit (i.e. Gitlab2Prov)

Application Class 3

- Development team
- User community
- External usage
- Long-time relevance
- High risk
- Example: High-end simulation code (i.e. Tau)

Guidelines address risk minimisation

Application classes: Example progression
Testing

Class 0	Class 1	Class 2	Class 3
	+ Basic operational testing	+ Test strategy	+ Quality compliance checks
	+ Repo contains test artifacts	+ Systematic functional testing	+ Regression tests
		+ Basic metrics trends (results, coverage, style)	+ Non-deterministic functional testing
			+ Test metrics defined, recorded, analysed
			+ Trends of new errors investigated

Example
Application Class 1

Reusable Python script for data analysis

- Small tool, >1 internal users

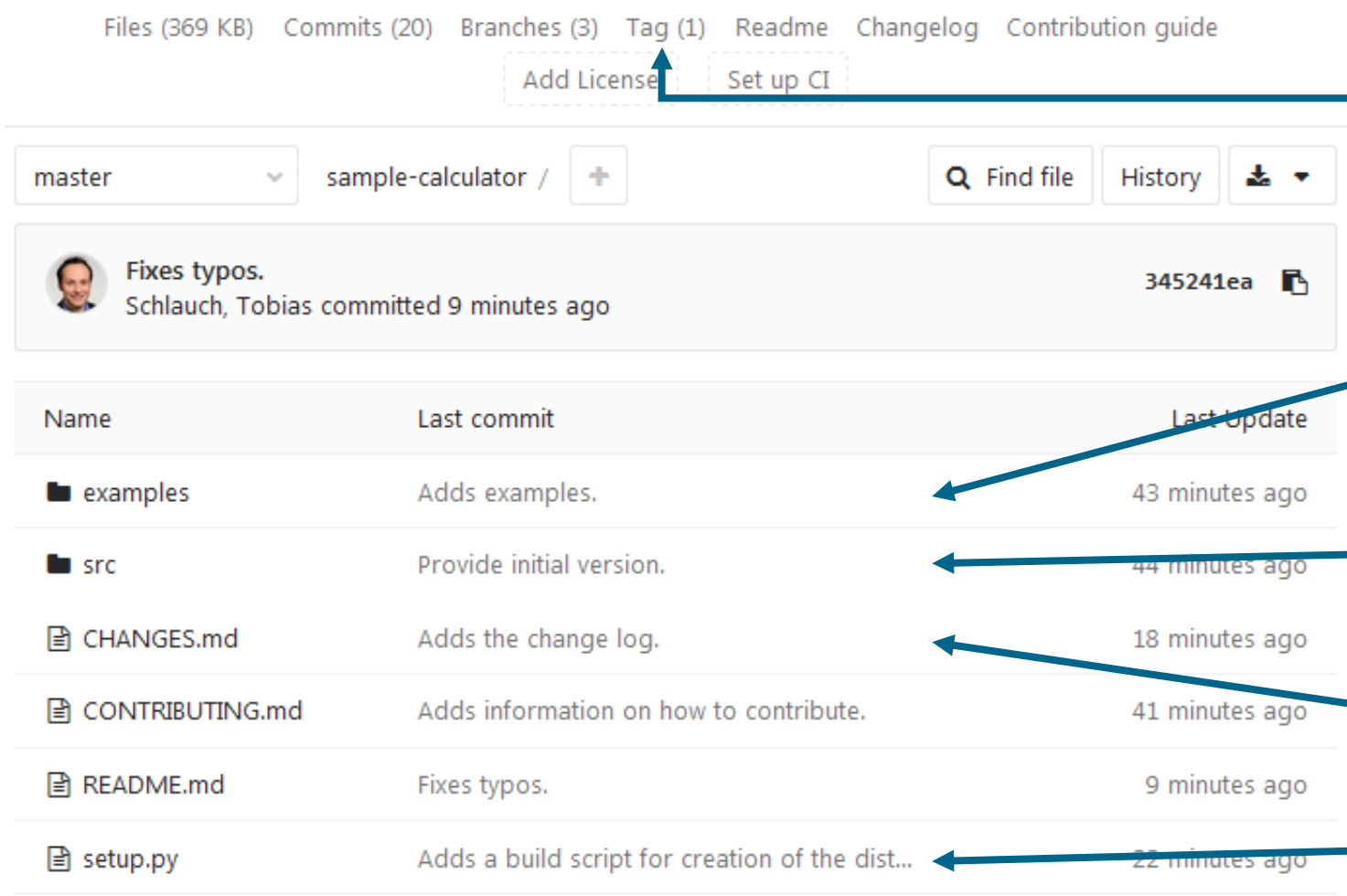
Generic recommendations for Application Class 1 apply:

- Apply **version control**
- Apply **generic coding style**
- Basic **modularization**
- **Avoid code duplication** and **over-engineering**
- **Automated build** of executable version
- **Essential documentation**
- **Testing** and **release tagging** for internal releases
- **Consult DLR Open Source Guidelines** for public releases

Recommendations have to be mapped into the actual development context.

Example implementation overview
Application Class 1

Git repository on collaborative development platform



- Releases correspond to tags
- Release package download

- Examples provide reference input values and results

- Code modularization in functions
- Coding style applied

- Documents changes (≈ changelog)

- Build script for packaging/installation
- Release tags (semantic versioning)

Example implementation documentation
Application Class 1

README.md: main documentation – CONTRIBUTING.md: contributor information (not shown)

What is SampleCalculator?

SampleCalculator is a command line tool to calculate characteristic values of a sample.

It provides the following features:

- Reading sample values from command line and CSV (Colon Separated Values) files.
- Calculation of average, variance, and standard deviation.
- Configurable logging of results and interim results.
- Easy integration of new input sources
- Extensible by easily adding new calculations

SampleCalculator targets **scientists** who want to easily perform such calculations as part of their workflow and **Python developers** who want to integrate the functionalities into their software. We implemented as we have not found a suitable, zero-dependency alternative.

> The current version is only an initial alpha version which is **NOT** suited for production use. Particularily, it is not sufficiently tested with large data sets. It requires **Python** >= **3.4** and has been only tested on **Windows 7** so far. However, it should basically work on operating system.

- Explanation of software purpose
- Overview of main features
- Important usage constraints and conditions

How can I install it?

- Make sure that you use Python >= 3.4
- Download the latest package
- Extract it to a directory

- Basic installation and usage information
- Roadmap
- …

Conclusion

- **Context & purpose**
  - Institutional software development guidelines
  - Define measures to safeguard appropriate software quality
  - Enable use case-sensitive development of sustainable research software
- **Genericity, building on minimum requirements**
- **Mobility between application classes**
- **Criteria:**
  - Criticality
  - Expected scope/user base
  - Expected lifetime in maintenance
  - Distribution mode (none, internal, public)

Questions?

- rse.dlr.de
- stephan.druskat@dlr.de
- @sdruskat@fosstodon.org

- DLR Software Engineering Guidelines:
  - EN: HTML, PDF
  - DE: HTML, PDF
  - T. Schlauch, M. Meinelund C. Haupt, „DLR Software Engineering Guidelines“. Zenodo, Aug. 17, 2018. doi: 10.5281/zenodo.1344612.