SUB RDD Technical Reference

21. Dezember 2018

<https://github.com/subugoe/rdd-technical-reference>

Table of Contents

1 Purpose

This guideline should help you getting started with a new software development project (or improving an existing one!) in the Research and Development Department of the Göttingen State and University Library.

Our goal is to establish better software quality by following standards the developer team has mutually agreed upon. Roughly basing on the EURISE-Network Technical Reference, these standards are discussed, worked out and decided on in the Software Quality Working Group which meets biweekly on Tuesdays at 12:30-13:30. However, they aren't cast in stone, so in case you have a good idea for a better standard, feel free to contribute!

2 Status

This document is a living document and will be extended every time the Software Quality Working Group has agreed upon a new standard for software projects in RDD.

3 Obligatoriness

Every developer in RDD should stick to these guidelines. Of course you can adjust them to the project specific requirements after consulting your PI.

4 Guidelines

4.1 Do you stick to our code style guides?

4.1.1 General

The basic definitions are given by our EditorConfig, i.e. Unix line breaks and 2 space indentation.

4.1.2 Specific for programming languagues

For the more prominent programming languages we have formatting and general style guides we ask you to follow:

- **Java**: The Java style guide can be found here. It's based on the Google style guide for Java with some minor RDD specific setting. You can configure Eclipse to use it automatically at *Eclipse > Preferences > Java > Code Style > Formatter.* Just load the RDD Eclipse Java Google Style in the formatter preferences and use it in your RDD projects.

- **JavaScript**: Airbnb JavaScript Style Guide. @TODO uv: Tutorials verlinken!
- **HTML/CSS**: Google HTML/CSS Style Guide.
- **XQuery**: xqdoc style guide with the following addenda:
  - use double quotes instead of single quotes (for easy escaping)
  - use four spaces for a TAB
- **XSLT**: Since there is no official style guide for XSLT, we decided to write our own, resulting from common best practices and own experiences within the department.
- **SPARQL**: For SPARQL there is not really any official style guide and there is no possibility to simply include any code style automatically using a code style file. We just collect some advices how to format and use SPARQL code.
  - declaration of variables should start with a **?** (and not with a **$**).
  - opening parenthesis **{** should be at the end of the line. Closing parenthesis in a separate line.

```
SELECT * WHERE {
    ?s ?p ?o .
} LIMIT 10

- group concatenations in SELECT command should be in seperate lines.

    **TODO** @Max: Provide example
```

4.2 Is your software fully documented?

4.2.1 General issues

- don't document computer language's interna
- best use language structure to document
- write the best documentation you can
- documentation and variable language is American English
- should be as close to the code as possible
- every code repository must provide
  - a README.md file that contains

    * link to original repository
    * short introduction
    * link to demo instance
    * example or demo installation
    * link to licence file
    * contribution guide
    * link to style guide
    * link to bugtracker/project managemenmt system
    * known issues
    * badges to ci status

A good example for a README structure can be found here.

  - a LICENCE file

4.2.2 Developer documentation

Software architecture Each software project should be documented by using an architecture diagram that helps understanding its basic functionality. Using tools to generate diagrams such as UML class diagrams might not to be possible or useful in every case.

Examples:

- Generating call graphs in eXist-db
- @TODO fu: Looks for JAVA UML things (crud?)

Call diagrams/graphs can be useful to help comprehending code and service calls. They should exist for every API method.

API documentation

- used parameters, author and since annotations
- example for Java: `https://lab.sub.uni-goettingen.de/self-updating-docs.html`
- links to callers? who is calling this method, and when?
- meet and write documentation together regularly (documentation sprint)?

4.2.3 Admin Documentation

- how to install the software, how to run and/or restart it, how to test the installation, . . .
- server documentation

4.2.4 User Documentation

- how to use the software and APIs, FAQs, walkthroughs, ...
- guided tour (Bootstrap Tour) as user documentation
  - for SADE portal usage (such as Fontane, BdN, Architrave)
  - for complex Digital Editions
- screencasts

4.3 Which version control do you use? You do use version control, don't you?

We are using GIT in RDD! Nothing else!

We recommend to use the gitflow Workflow, especially when several developers contribute to your software. When using gitflow you should protect the develop and master branches on your server to avoid pushing to them by accident.

In RDD we use several Git repository servers:

- projects.gwdg.de
- GitLab
- GitHub

Which one is most suitable for your project depends on several factors including the project itself, existing code, which CI/CD solutions you want to use, ... Don't hesitate to consult the developer team in case you have questions!

Closing issues automatically via commit message(s) is recommended; The exact way to do it depends on your Git repository server. Issues can also be referenced across repositories (cf. link).

We have got an RDD team on Github: `https://github.com/orgs/subugoe/teams/fe`

Consider mirroring of repos for project visibility (e.g. mirror Gitlab/Projects code to Github?)

4.4 Are you tracking your bugs properly?

A bug tracking system is mandatory! Please use the respective bug tracking system of your repo and/or project management solution.

4.5 Are you testing your code?

We aim to have a test coverage of **100%** (except for getter and setter methods). Whether you achive this by Test Driven Development (TDD) or not is specific to your preferred way to work.

Please keep in mind not only to write a test for each of your functions but also to consider all possible outcomes. It is e.g. not sufficient to test if a function creates a file if the written content depends on variables etc.

Examples for different programming languages are:

- **XQuery**: `https://gist.github.com/joewiz/fa32be80749a69fcb8da`

4.6 Code building and continuous integration

- @TODO: Code building (such as Jenkins, Gitlab Runner)
- @TODO: Provide complete examples for Jenkins and GitLab runner.
- @TDOO: Monitoring (such as Icinga, Metrics)

5 Code quality level for RDD

- Evaluate Software maturity Levels from CESSDA: **TODO**: @mw
- @TODO: Code reviewing, evaluate quality level
- @TODO: Wissenschaftliche Standards für wissenschaftliche Software?!

6 Licencing

- clarify software licence before programming with your PI
- add licence to code header
- @TODO: depends on used software libraries, project and/or funder

7 Retirement of software

- clarify if software is no longer supported

8 Helpful links and references

- eurise-network technical-reference: `https://github.com/eurise-network/technical-reference`

- DHTech – An international grass-roots community of Digital Humanities software engineers: `https://dh-tech.github.io`
- The Software Sustainability Institute, Guidelines and publicartions: `https://www.software.ac.uk`
- The Joel Test: 12 Steps to Better Code: `https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code`
- Software Quality Guidelines: `https://github.com/CLARIAH/software-quality-guideli`
- Software Testing Levels: `http://softwaretestingfundamentals.com/software-testing-levels`

9 Glossary