

3. Juni 2019 <<https://github.com/subugoe/rdd-technical-reference>>

Table of Contents

1	SUB RDD Technical Reference	3
1.1	Guidelines	3
1.1.1	General	3
1.1.2	Specific for programming languages	3
1.2	Is your software fully documented?	5
1.2.1	General issues	5
1.2.2	Developer documentation	5
1.2.3	Admin Documentation	6
1.2.4	User Documentation	6
1.3	Which version control do you use? You do use version control, do you?	6
1.4	Are you tracking your bugs properly?	7
1.5	What is your test coverage?	7
1.6	Building code and continuous integration	8
1.6.1	Building code	8
1.6.2	Continuous integration	8
1.7	Deployment and maintenance	12
2	Code quality level for RDD	12
3	Licencing	12
4	Retirement of software	12
5	Helpful links and references	12

1 SUB RDD Technical Reference

Author: Software Quality Working Group

Purpose: This guideline should help you getting started 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 DARIAH Technical Reference, these standards are discussed, worked out, and decided 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!

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

1.1 Guidelines

Do you stick to our code style guides?

1.1.1 General

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

Unfortunately, not all editors support EditorConfig. In case you use **eXide**, the IDE that comes with exist-db, you can set 2 space indentation as default by editing `/db/apps/eXide/src/preferences.js`.

1.1.2 Specific for programming languages

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:** For JS we use the Airbnb JavaScript Style Guide.
- **HTML/CSS:** For HTML/CSS we agreed upon the Google HTML/CSS Style Guide.
- **XQuery:** We use the xqdoc style guide with the following addenda:

- use double quotes instead of single quotes (for easy escaping)
- use four spaces for a TAB (because eXide makes it so)
- **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.
- **Python**: For Python PEP 8 should be used, Django has a styleguide based on PEP-8 with some exceptions: Django-Styleguide. There are linters and tools like flake-8 and pep-8 available to support.
- **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 separate lines.”

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dct: <http://purl.org/dc/terms/>
```

```
SELECT DISTINCT
(group_concat( distinct ?conceptName;separator="; ") as ?conceptNames)
(group_concat( distinct ?conceptUri;separator="; ") as ?conceptUris)
(group_concat( distinct ?next;separator="; ") as ?nexts)
(group_concat( distinct ?def;separator="; ") as ?defs)
WHERE {
    <' + uri + '> skos:narrower ?conceptUri.
    ?conceptUri skos:prefLabel ?conceptName.
    OPTIONAL{?conceptUri skos:narrower ?next.}
    OPTIONAL { ?conceptUri skos:definition ?def.}
    FILTER(LANG(?conceptName) = "" || LANGMATCHES(LANG(?conceptName), "en"))
}GROUP BY ?conceptUri
```

1.2 Is your software fully documented?

1.2.1 General issues

- don't document computer language's internal
- best use language structure to document
- write the best documentation you can
- documentation and variable language is American English
- should be as code-near as possible
- every code repo must have
 - 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 management system
 - * known issues
 - * badges to ci status

A good example can be found here.

- a LICENCE file

1.2.2 Developer documentation

Architecture of the software Each software project should be documented using an architecture diagram that helps understanding its basic functionality (using tools to generate diagrams such as UML class diagrams seems not to be possible in every case).

Examples:

- Generating call graphs in eXist-db

Call diagrams can be useful to follow code and service calls and should be existing 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 must not be listed in the documentation, because this info will be deprecated soon. Strongly recommended is using call stacks of tools like Eclipse (Java) and/or Call Graph Module (SADE).
- We do meet and write documentation together regularly (documentation sprint) every friday from 1 PM in the RDD meeting room. WE NEED COOKIES!
- Document REST-APIs using openAPI if possible. OpenAPI docs should be located at /doc/api on servers.

1.2.3 Admin Documentation

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

1.2.4 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

1.3 Which version control do you use? You do use version control, do you?

We are using GIT in RDD! Nothing else! How it works, please see <https://git-scm.com/doc>.

We recommend to use Gitflow Workflow: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>, Cheat Sheet: <https://danielkummer.github.io/git-flow-cheatsheet>), if possible on server side: use protection of the develop and master branches. All specific branches working on an issue described in a bug tracker may utilize

the following naming scheme: `[track]/#[ISSUENUMBER]-[KEYWORD]`, e.g. `bugfix/#12-flux-capacitor`.

A github workflow used in DARIAH-DE and related services is described in the DARIAH-DE Wiki: <https://wiki.de.dariah.eu/display/DARIAH3/DARIAH-DE+Release+Management#DARIAH-DEReleaseManagement-Beispielmitdevelop-undm>

Automatically closing issues via commit message depends on the Git repository server. Issues can also be referenced across repositories (cf. link).

Which repo you are using depends on:

- the project
- existing code
- using Gitlab Runners
- ...

We use the following at the moment in RDD:

- Projects (GWDG) -> <https://projects.gwdg.de>
- Gitlab (GWDG) -> <https://gitlab.gwdg.de>
- github.org -> <https://github.com/subugoe>

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?)

1.4 Are you tracking your bugs properly?

A bug tracking system is obligatory! Please use the respective bug tracking system of your repo and/or project management solution (please see chapter version control)!

1.5 What is your test coverage?

We aim to have a test coverage of **100%** (except for getter and setter methods). Whether you achieve 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>

1.6 Building code and continuous integration

1.6.1 Building code

The reason for using a build tool is to be able to build and/or test a code project with one command (after checking out). Another reason is to include dependency management.

Build tools we are using at the moment

- **bash scripting:** (BdN Print, Fontane Print)
- **eXist:** Ant (SADE)
- **Java:** Maven (TextGrid)
- **JavaScript:**
 - bower (DARIAH-DE GeoBrowser, tgForms)
 - cake (tgForms)
 - NPM (DARIAH-DE Publikator, tgForms)
 - rake (DARIAH-DE GeoBrowser)
- **Phython:**
 - make (Sphinx documentation)
 - PIP (DiscussData)
- **Ruby:** bundler (DARIAH status page)

Build tools we want to evaluate

- gradle

1.6.2 Continuous integration

We want to use CI as soon as possible in new projects.

The workflows we are using currently in Jenkins and Gitlab Runner are:

- Code building
- Testing
- Code analyzer (Sonar)
- Packaging (JAR, WAR, DEB, XAR)
- Distribution (Nexus, APTLY repo, eXist repo)
- Release Management (@TODO: where to put this? gitflow?)

Sample configuration of the GitLab Runner The following example illustrates how the GitLab Runner is used in SADE. The fully documented version of this file can be viewed [here](#).

```
image: docker.gitlab.gwdg.de/fontane-notizbuecher/build:latest
```

```
stages:
  - build
  - test
  - deploy
```

```
build-develop:
  except:
    - master
    - tags
  stage: build
  script:
    - ant test
  artifacts:
    paths:
      - build/*.xar
      - test/
```

```
build-master:
  only:
    - master
  stage: build
  script:
    - cp master.build.properties local.build.properties
    - ant test
  artifacts:
    paths:
      - build/*.xar
      - test/
```

```
installation:
  except:
    - tags
  stage: test
  script:
    - bash test/eXist-db-*/bin/startup.sh | tee output.log &
    # wait for eXist to have started
    - while [ $(curl --head --silent http://localhost:8080 | grep -c "200 OK") == 0 ]; do
    # shutdown eXist
```

```

- bash test/eXist-db-*/bin/shutdown.sh
- ls -al /tmp; mv /tmp/tests-* . || true
artifacts:
  paths:
    - output.log
    - test/tests-*.xml
    - test/eXist-db-*/webapp/WEB-INF/logs/expath-repo.log
  # this enables us to get information like test coverage.
  reports:
    junit: test/tests-*.xml

upload:
  only:
    - master
    - develop
  except:
    - tags
  stage: deploy
  script:
    - FILENAME=$(ls build/*.xar)
    - curl -u ci:${EXIST_UPLOAD_PW} -X POST -F file=@${FILENAME} https://ci.de.

```

Sample configuration of the Jenkins CI (Multibranch Pipelines)

- On commit and push to the <https://projects.gwdg.de> gitolite repo (such as <https://projects.gwdg.de/projects/tg-crud/repository>) Jenkins on ci.de.dariah.eu <https://ci.de.dariah.eu/jenkins> is notified (see projects' gitolite configuration)
- Jenkins then is doing a checkout and build of configured branches (see Jenkins' project's multibranch pipeline configuration such as <https://ci.de.dariah.eu/jenkins/job/DARIAH-DE-CRUD-Services>. Here comes THE CRUD Jenkinsfile, as it is used for other projects in a very similar way:

```

#!/usr/bin/env groovy

node {
  def mvnHome

  stage('Preparation') {
    mvnHome = tool 'Maven 3.5.0'
    checkout scm
  }
}

```

```

stage('Build') {
    // We are deploying all JARs and WARs build here, SNAPSHOTS and RELEASES!
    sh "cd service && '${mvnHome}/bin/mvn' -U clean verify deploy -Pdhp.rep.deb"
}

stage('Publish') {
    def pom = readMavenPom file: 'service/pom.xml'
    def pVersion = pom.version
    def snapshot = pVersion.contains("SNAPSHOT")
    def tg = pVersion.contains("TG")
    def dh = pVersion.contains("DH")

    if (snapshot) {
        if (tg) {
            doDebSnapshot('tgcrud-webapp', 'service/tgcrud-webapp/target', pVersion)
            doDebSnapshot('tgcrud-webapp-public', 'service/tgcrud-webapp-public/target', pVersion)
        }
        if (dh) {
            doDebSnapshot('dhcrud-webapp', 'service/dhcrud-webapp/target', pVersion)
            doDebSnapshot('dhcrud-webapp-public', 'service/dhcrud-webapp-public/target', pVersion)
        }
    }
    else {
        if (tg) {
            doDebRelease('tgcrud-webapp', 'service/tgcrud-webapp/target', pVersion)
            doDebRelease('tgcrud-webapp-public', 'service/tgcrud-webapp-public/target', pVersion)
        }
        if (dh) {
            doDebRelease('dhcrud-webapp', 'service/dhcrud-webapp/target', pVersion)
            doDebRelease('dhcrud-webapp-public', 'service/dhcrud-webapp-public/target', pVersion)
        }
    }
}
}

```

- Stage *Preparation*: Prepare things
- Stage *Build*: Build, JAR, WAR, and DEB packages from source code, deploy JAR and WAR packages to the Nexus repo (at the moment <https://ci.de.dariah.eu/nexus> → in the near future <https://nexus.gwdg.de>). Jenkins is configured to deploy JARs and WARs via Maven and a Nexus deploy-account.
- Stage *Publish*: Publish DEB packages to the DARIAH-DE Aptly Repo

<https://ci.de.dariah.eu/aptly>. Jenkins is using a shared library of scripts and publishing is divided into four conditionals: TG version, DH version, SNAPSHOT version, or RELEASE version due to given version suffixes!

1.7 Deployment and maintenance

- @TODO: Puppet
- @TODO: Monitoring (such as Icinga, Metrics)

2 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?!

3 Licencing

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

4 Retirement of software

- clarify if software is no longer supported

5 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 Publications: <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-guidelines>

- Software Testing Levels: <http://softwaretestingfundamentals.com/software-testing-levels>
- Netherlands eScience Center Guide <https://guide.esciencecenter.nl>