SUB RDD Technical Reference

16. August 2019

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

Table of Contents

1 About this Document

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 EURISE Network Technical Reference, these standards are discussed, worked out, and decided in the Software Quality Working Group, which meets biweekly on Tuesdays at 13:00-14:00. 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. TODOs and addenda of this document are maintained here.

2 Style Guides

2.1 General

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

2.2 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 settings. 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 switching the preferences in eXide's setting isn't permanent)

- **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 style guide based on PEP-8 with some exceptions: Django-Styleguide. There are linters and tools like flake-8 and pep-8 available as 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 advice 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. Example:

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

'" - group concatenations in SELECT command should be in seperate 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
```

3   Is Your Software Fully Documented?

3.1   Docs Sprints

To ensure the best documentation of our code we can we meet on a weekly base for a code sprint to document everything we have coded throughout the week and haven't been able to document properly yet. The meeting takes place in the meeting room at 1pm. Cookies may be provided.

3.2   General

- don't document a language's specifics, e.g. operators

- best use language structure to document

- write the best documentation you can

- documentation and variable language is American English

- docs should be as close to the code as possible

- every code repository must have

    - a README.md file that contains

        * link to original repository (if the software is forked or otherwise based on preexisting software)
        * short introduction on what the repository is about
        * a guide how to get the software running (if makes sense)
        * link to demo instance
        * example or demo installation
        * link to license file
        * contribution guide
        * link to style guide
        * link to bug tracker/project management system
        * known issues
        * badges to CI status

    A good example can be found here.

    - a LICENCE file

3.3 Developer Documentation

3.3.1 Architecture of the Software

Each software project should be documented using an architecture diagram that helps understanding its basic functionality (even though using tools to generate diagrams such as UML class diagrams doesn't seem 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.

3.3.2 API Documentation

- The docs should comprise used parameters, author and @since annotations

- Example for Java

- Links to callers must not be listed in the documentation, because this info will be deprecated soon. It is strongly recommended to use call stacks of tools like Eclipse (Java) and/or Call Graph Module (SADE).

- Document REST-APIs using openAPI if possible. OpenAPI docs should be located at `/doc/api` on servers.

3.4 Admin Documentation

- The docs should comprise how to install the software, how to run and/or restart it, how to test the installation, . . .

- Server documentation: This type of documentation is provided and maintained in our wiki space. We have a template encompassing all information necessary: To create a wiki page for a new server navigate to the FE Server list, select ". . . " right beside the "Create" button and search for "FE-Server".

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

We exclusively use git in RDD. Please see `https://git-scm.com/doc` for information on how it works.

We recommend to use the Git flow Workflow (also consult the Cheat Sheet). For git flow it is safest to protect your master and develop branch on server side to avoid accidental pushes into these 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.

It is also recommended to automatically close issues via commit message; How this works exactly depends on the Git repository server. Issues can also be referenced across repositories (cf. link).

We use the following Git servers at the moment in RDD:

- Projects (GWDG) –> `https://projects.gwdg.de`

- GitLab (GWDG) –> `https://gitlab.gwdg.de`

- github.org –> `https://github.com/subugoe`

Which one is suitable for you depends on:

- the project you are working on

- existing code

- whether or not you want to use CI/CD or GitLab Runners

- . . .

We have got an RDD team on GitHub. Feel free to join us!

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

5 Do you track 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)!

6 Do you test your software?

We aim to have a test coverage of **100%** (except for getter and setter methods). This is understood on a component level, which means that every method should have at least one test. 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 writing tests in different programming languages are:

- **XQuery**

7 Building Code and Continuous Integration

7.1 Building Code

Ideally we use build tools to conveniently get a software running. 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**: (BdNPrint, FontanePrint)

- **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

7.2 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 (via GitLab Environments and gitflow)

7.2.1 Sample configuration of the GitLab Runner

The following example illustrates how the GitLab Runner is used in SADE.
The full and documented version of this file can be viewed here.

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

stages:
  - build

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/
```

7.2.2 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 is notified (see projects' gitolite configuration)

- Jenkins then does 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 -Pdhrep.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', pVe
      }
      if (dh) {
        doDebSnapshot('dhcrud-webapp', 'service/dhcrud-webapp/target', pVersion)
        doDebSnapshot('dhcrud-webapp-public', 'service/dhcrud-webapp-public/target', pVe
      }
    }
    else {
      if (tg) {
        doDebRelease('tgcrud-webapp', 'service/tgcrud-webapp/target', pVersion)
        doDebRelease('tgcrud-webapp-public', 'service/tgcrud-webapp-public/target', pVer
      }
      if (dh) {
        doDebRelease('dhcrud-webapp', 'service/dhcrud-webapp/target', pVersion)
        doDebRelease('dhcrud-webapp-public', 'service/dhcrud-webapp-public/target', pVer
      }
    }
  }
}
```

- Stage *Preparation*: Prepare things

- Stage *Build*: Build, JAR, WAR, and DEB packages from source code,
  deploy JAR and WAR packages to the Nexus repo. For further infor-
  mation on Nexus cf. ci's server documentation. 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.
  Jenkins is using a shared library of scripts and publishing is devided
  into four conditionals: TG version, DH version, SNAPSHOT version,
  or RELEASE version due to given version suffixes!

11

8 Deployment and maintenance

8.1 Puppet

8.2 Monitoring

8.3 Release Management

9 Code quality level for RDD

9.1 Code review

We want to ensure code review for all major commits, in gitflow for everything that is subject to be merged into `develop`.

For projects with more than one developer in the team it is preferred to have code reviews within the team, in other cases your friendly RDD developer team is on your side.

9.1.1 Proof of concept

When preparing a proof of concept that is always labeled `poc`, a code review is not necessary.

10 Licensing

- clarify software license before programming

- add license to code header

Best practice is to maintain a file listing all third-party packages that are part of the software. This list should hold the following metadata and SHOULD be prepared like the table below, always in alphanumeric order.

```
| name | license | origin |
|---|---|---|---|
| foo | barware | github.com/foo/bar |
```

Maybe the `license-maven` plugin will help you.

11 Retirement of software

- clarify if software is no longer supported

12 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`