Best Practices in Research Data Management

Hauke Sonnenberg, Michael Rustler & Christoph Sprenger

2019-09-24 12:52:03

Contents

Preface

> "These days, data trails are often a morass of separate data and results and code files in which no one knows which results were derived from which raw data using which code files."
>
> — Professor Charles Randy Gallistel, Rutgers University

This document is the outcome of the KWB project FAKIN (Forschungsdatenmanagement an kleinen Instituten = research data management at small institutes).

This report defines Best Practices for research data management especially designed for small reserach institutes.

The document is outlined as follows:

- Chapter 1 explains why and for whom research data management is important
- Chapter 2 defines Best Practices for different topics. In section Data workflow of this chapter we propose to distinguish between raw data, processed data and project results.
- Chapter 3 gives an overview on three case studies with different data related tasks that have been solved within the FAKIN project, while
- Chapter 4 what data challenges occured in the past in other KWB projects and how they were solved in order to increase our awareness at KWB,
- Chapter 5 provides FAQs and finally
- Chapter 6 is a glossary for explaining commonly used terms within this document.

This document is assumed to be a "living" document. We highly appreciate any comments and suggestions for improvements. What are your experiences with research data managment tasks? Can you provide solutions for specific tasks?

Chapter 1

Introduction

1.1 What is Research Data Management?

Research data management comprises all parts of the "life cycle" of data.

It "is part of the research process, and aims to make the research process as efficient as possible, and meet expectations and requirements of the university, research funders, and legislation." (https://www2.le.ac.uk/services/research-data/rdm/what-is-rdm)

1.2 Why is Research Data Management important?

According to Labfolder.com (2018) research data management (RDM) is important as it:

- Saves time for research,
- Avoids risk of data loss,
- Ensures transparency and reproducibility,
- Increases data visibility and number of citations,
- Fulfills funders' requirements and receives more grants,
- Produces new knowledge and makes more discoveries just by re-using data,
- Archives, retrieves and re-uses own data.

1.3 Why RDM for small institutes?

This document focuses on research data management at small institues. In terms of data management, we found the following characteristics of small institutes:

The transfer of knowledge is particularly important because the loss of knowledge can be considerable in case that an employee leaves the institute.

Small institutes do not have an IT department. They do not have employees that are dedicated to data managment tasks only. Often, there are no data management guidelines defined. Data management is not centrally organised but is left to the different project leaders and researchers. All employees are expected to be their own data experts. Depending on the individual skills of the different employees, different levels of data handling practices are applied.

In small institutes, work is organised in terms of projects. There may not be an overall strategy or target that is followed by the institute. Targets are depending on research programmes and requirements may be set by funding organisations. The projects may be very different in terms of amount and diversity of data.

Small institutes have a simple organisational structure. This may allow for a simple informal exchange of information. This can lead to fast answers and decisions but very often has the drawback that processes are not documented sufficiently. In small institutes with a flat hierarchy innovations such as the usage of new methods or tools very often start as initiatives of single employees or are first applied in single projects. Practices that have been proven to be beneficial may then be used in future projects or even be set as a standard for the whole institute (bottom-up). It is less probable that the head of the institute sets guidelines to be followed consistently in all projects (top-down).

The aim of this document is to define best research data managment practices that are assumed to be particular useful for small institutes.

1.4 Recommended literature

A very good book on how research data management (RDM) improves reproducibility is provided by Kitzes et al. (2018) and available online at: http://practicereproducibleresearch.org/.

- Introduction to research data management in the geosciences (in German!)
- Handbook research data management (in German!) (Buettner et al., 2011)

Chapter 2

Best Practices

2.1 Plan and Fund a New Project

Before starting with a new project, perform the following actions that are detailed in the following sections:

- Choose a project identifier and identifiers for the project partners and organisations that data are expected to be received from.
- Create a data management plan (DMP) for the project.
- Determine a "data curator" for the project.
- Setup the folder structures for the project.

At the start of a research project

- Create a subfolder for your project and subfolders for the organisations in the rawdata folder structure.

At the start of a project or if an employee or trainee enters the project

- Give an introduction to our research data management as described in this document.

Regularly during the project

- Check if the folder structure within your project's rawdata subfolder still complies with the rawdata folder structure and clean the structure, if not.

2.1.1 Project and owner identifiers

Documents and data that are collected or created during the lifetime of a project will be stored in the form of electronic files on the institute's file server or on the

personal computers of the employees. We aim at being able to clearly identify the related project and the owner of the data for any file or folder that resides on the institute's file system.

Therefore, we recommend to use unique identifiers, either as (part of) the name of the file or as (part of) the name of the folder or parent folder that the file resides in. The usage of a consistent project identifier becomes particularly important when, as we suggest for a good data workflow, different types of data and documents of the project are stored at different places on the file system.

Together with her team, the project leader is requested to define the project identifier at the start of the project. Whenever the belonging of a file or folder to the project is to be indicated, only this "official" identifier in the exact spelling that was defined, is to be used.

The project identifier should

- be unique within all identifiers of current or former projects that have ever been carried out at the institute,
- be as short as possible but as long as required to be meaningful and easy to remember.

In order to minimise the risk of spelling mistakes and to avoid errors in automated processing we require the identifier to

- start with a lowercase letter,
- consist of only lowercase letters (`a-z`), digits (`0-9`) or the hyphen (`-`),
- end with a lowercase letter or digit,
- consist of at least four and at last 16 letters,
- not contain two hyphens in sequence (`--`).

Examples for valid project identifiers are:

- sema-berlin-2
- aquanes
- ist4r

Invalid project identifiers are:

- sema-berlin_2 (has underscore)
- Aquanes (has uppercase letters)
- 4you (starts with a digit)
- abc (does not have at least four letters)
- this-project-name-is-too-long (has more than 16 letters)

The project identifier is not to be confused with the project acronym or project title as it has been aggreed on by the project consortium during the project development.

We propose to document the choice of project identifier, project acronym and project title in a project's metadata file. This will be described in another chapter of this document.

For data that are received from external partners or organisations we recommend to indicate the origin of the data by means of another type of identifier in the names of the corresponding files and folders. The same strict naming convention as for the project identifiers should be applied to these institutional identifiers that are to be documented in another special metadata file. At the start of a project it should be checked if the organisations that are expected to receive data from are listed in this metadata file. If not, the metadata file is to be extended as necessary.

2.1.2 Data Management Plan

Look through a (Research) Data Management Planning (DMP) checklist, e.g. here: DMP checklist

If the project's funding organisation demands a Data Management Plan, use the template that is provided by the funder. You find different DMP-templates (e.g. that required for European Horizon 2020 projects) here: DMP Online.

Have a look at the online tool RDMOrganizer that lets you describe the data-related aspects of your project by letting you answer questions.

2.1.3 Responsibilities of the Data Curator

The data curator is responsible for:

- saving raw data in the `rawdata` network folder (see below). The curator will get a special login account with write-access to the `rawdata` folder. The login information can be got from Michael Rustler or Hauke Sonnenberg, the members of the FAKIN project team.
- regularly checking if both the file and folder names as well as the folder structure comply with the best practices described in this document and with the naming conventions aggreed on at the start of the project. The curator checks if metadata files are available and up-to-date.

2.1.4 Setup folder structures

At the start of the project, define and create folder structures for the three areas `rawdata`,`processing`, `results` that we propose in this document.

Create the folders and provide a metadata file (see below) describing their meaning. In addition to the general recommendations given in this document, create and document naming conventions for files and folders for your project.

2.1.5 Electronic Lab Notebooks

Electronic Lab Notebooks (ELNs) https://www.labfolder.com/research-data-management/ ELNs are an essential asset for researchers to fulfil any requirements for data management, and they create direct bridges between scientists and stakeholders. By adopting an ELN, the data lifecycle can proceed smoothly and easily: from creating and collecting data digitally in one place to one-click data archiving, ELNs empower researchers by allowing them to implement their RDM plan without effort and time investment.

2.2 Data Storage

A logical and consistent folder structure as well as naming conventions and versioning rules for your data files help you and your colleagues find and use the data. The following rules will save time on the long run and will help avoid data duplication.

This chapter describes

- where to store data files, see Where to Store Data,
- what names to choose for files and folders, see Naming of Files and Folders,
- how to group files in folders, see Folder Structures.

With the following recommendations we intend to comply with Hart et al. (2016).

2.2.1 Where to Store Data

2.2.1.1 Network instead of local hard drives

In general, data related to one project should be accessible to all the persons that are involved in the project. It should be avoided that project relevant files are permanently stored on local hard drives or on personal network drives. Project data should therefore be stored on network drives to which all project members have access.

2.2.1.2 The Input-Processing-Output (IPO) Model

A research institute aims at creating knowledge from data. A typical data flow may look like the following: raw data are received or collected from foreign sources or created from own measurements. The raw data are processed, i.e. cleaned, aggregated, filtered and analysed. Finally result data sets are composed from which conclusions are drawn and published. To cut it short: raw data get processed and become result data.

In computer science, the clear distinction between data input, data processing and data output is a well-known and widely-used model to describe the structure of an information processing program. It is referred to as the input-process-output (IPO) model. We recommend to apply this model to the distinction of different categories of data:

- `raw data` (= input data),
- `data in processing` (=data processing) and
- `result data` (= output data).

2.2.1.2.0.1 Why to use the IPO Model?

Using the IPO model:

- Minimizes the risk of overwriting, deleting of files and folders by automatic data processing (e.g. scripts).
- Rawdata are protected against accidental overwriting.
- Helps to keep files and folders clearly organised.
- Reflects roles and responsibilities of different project team members (e.g. # project manager is mainly interested in results).
- Helps avoid deep folder structures.

2.2.1.2.0.2 IPO Model in Practice

According to the three categories we suggest to create three different areas, represented by three different network drives, on the top level of your file server. The first area is for raw data, the second area is for data in processing and the third area is for result data. Within each area, the data are organised by project first, i.e. each project is represented by one folder within each of the network drives:

```
//server/rawdata
    project-1/
    project-2/
    …

//server/processing
    project-1/
    project-2/
    …

//server/results
    project-1/
    project-2/
```

...

Sub-folder structure within the project folders in each of these top-level network drives is described in the section ([link]).

2.2.1.3 Raw data

As raw data we define data that we receive from a measurement device, project partner or other external sources (e.g. internet download) even if these data were processed externally. Raw data are, for example, analytical measurements from laboratories, filled out questionnaires from project partners, meteorological data from other institutes, measurements from loggers or sensors, or scans of hand written manual sampling protocols. Especially in environmental sciences, raw data often cannot or only with high costs be reproduced (e.g. rainfall, river discharge measurements). They are therefore of high value. Raw data are often large in terms of file size or number (e.g. measurements of devices logging at high temporal resolution). Raw data can already come in a complex, deep folder structure. Raw data are closely related to metadata such as sensor configurations generated by loggers or email correspondence when receiving data by email from external partners. We acknowledge the high value of raw data by storing them on a dedicated, protected space and by requiring them to be accompanied by metadata ([link] Metadata).

Raw data are stored in an unmodified state. All modifications of the data are to be done on a copy of the raw data in the "processing" space (see next). The only modification allowed is the renaming of a raw data file, given that the renaming is documented in the metadata. Once stored, raw data are to be protected from being accidentally deleted or modified. This is achieved by making the raw data space write-protected ([link] FAQ: How to make a file write protected).

2.2.1.4 Data in Processing

As "data in processing" we understand data in any stage of processing between "raw" and "final", i.e. all intermediate results, such as different stages of data cleaning, different levels of data aggregation or different types of data visualisation.

We recommend to store data in these stages in their own space on the file system. This space is assumed to be a "playground", where the researchers are asked to store all these intermediate results. This space is where different approaches or models or scenarios can be tested and where, as a result, different "versions" of data are available. The data processing space is intended to be used for data only, and not for e.g. documents, presentations, or images.

Compared to the raw data space the data processing space is expected to require much more disk space.

- stages of data cleaning, different levels of data aggregation or
- types of data visualisation. The data processing area is where
- approaches or models or scenarios can be tested and where, as a result,
- "versions" of data are available.

Compared to the raw data network drive the data processing network drive is expected to require much more disk space.

2.2.1.5 Result Data

With "result data" we mean clean, aggregated, well formatted data sets. Result data sets are the basis for interpretation or assessment and for the formulation of research findings. We consider all data that are relevant for the reporting of project results as result data. Result data will very often be spreadsheet data but they can also comprise other types of data such as figures or diagrams. We propose to prepare result data in the data processing area (see above) and to put symbolic links that point to the corresponding locations in the data processing folder into the result data folder. The idea is that the result area always gives the view onto the "best available" (intermediate) project results at a given point in time. Using symbolic links instead of file copies avoids accidental modification of data in the result data area that are actually expected to happen in the data processing area.

Often, result data sets are the result of temporal aggregation. They are consequently smaller in size than raw data sets. There will also be less result data sets than there are data sets representing different stages of data processing. For these reasons, the result data space is expected to require much less disk space than the spaces dedicated to raw data and data in processing.

2.2.1.6 Clean Datasets

In a project-driven research institute, almost all data processing steps are closely related to their specific research project. One project may e.g. require to prepare rain data for being fed into a rainfall-runoff and sewer simulation software. Unprocessed (raw) rain data are received from a rain gauge station and cleaned. The clean rain data are then converted to the format that is required by the sewer simulation software.

In this example, the clean rain data are a data processing output. They are also the input to further processing and thus the source of even more valuable results.

The specific rain data file that is input to the sewer modelling software is the final (rain data) result. However, the clean rain data that are an intermediate result in the context of one project are themselves already valuable results. They

can be used in other projects that require clean rain data for other purposes, e.g. for looking at climate change effects.

We recommend to store clean datasets in their own space. In this space the datasets are organised by topic, not by project:

```
//server/treasure
  rain/
  flow/
  level/
```

This increases the visibility of existing clean datasets and reduces the risk that work that has already been done in one project is done again in another project. Often, people start again from the raw data even though somebody already cleaned that data.

2.2.2 Naming of Files and Folders

A concise and meaningful name of your file and folder is the key to relocate your data. Whenever you have the freedom to name your data file and structure your project folder you should do so. Names should be concise and meaningful to you and your colleagues. Your colleague, who may not be familiar with the project, should be able to guess the content of a folder or a file by intuition. Naming conventions are also necessary to avoid read errors during automatic data processing and to prevent errors when working on different operating systems.

Please comply with the following rules:

Rule A: Allowed Characters

The following **set of characters** are authorized in file or folder names:

- upper case letters `A-Z`,
- lower case letters `a-z`,
- numbers `0-9`,
- underscore `_`,
- hyphen `-`,
- dot `.`

If you want to know why some characters are not authorized, please check the FAQ:

- Why are special characters not allowed?
- Why is the space character not allowed?

Instead of German umlauts and the sharp s (ä, ö, ü, Ä, Ö, Ü, ß) use the following substitutions: ae, oe, ue, Ae, Oe, Ue, ss.

Rule B: Separation of Words or Parts of Words

Please use the characters underscore _ or hyphen - instead of space. Use underscore _ to separate words that contain different types of information:

- results_today instead of results-today
- protocol_hauke instead of protocol-hauke

Use hyphen - instead of underscore _ to visually separate the parts of compound words or names:

- site-1 instead of site_1,
- dissolved-oxygen instead of dissolved_oxygen,
- clean-data instead of clean_data.

Use hyphen - (or no separation at all) in dates (i.e. 2018-07-02 or 20180702).

Using hyphen instead of underscore in composed words will not split the composed words into their parts when splitting a file or folder name at underscore.

For example, splitting the name project-report_example-project-1_v1.0_2018-07-02 at underscore results in the following words (giving different types of information on the file or folder)

- project-report (type of document),
- example-project-1 (name of related project),
- v1.0 (version number),
- 2018-07-02 (version date).

Rule C: Capitalisation

From the pure data management's point of view it would be best not to use upper case letters in file or folder names at all. This would avoid possible conflicts when exchanging files between operating systems that either care about case in file names (as e.g. Unix systems) or not (as e.g. Windows systems).

If allowing upper case letters it should be decided on if and when to use capitals. Having a corresponding rule in place, only one of the following spellings would, for example, be allowed:

- dissolved-oxygen (all lower case),
- dissolved-Oxygen (attributes lower case, nouns upper case),
- Dissolved-oxygen (first letter upper case),

- `Dissolved-Oxygen` (all parts of compound words upper case).

Rule D: Avoid Long Names

At least on Windows operating systems, very long file paths can cause trouble. When copying or moving a file to a target path that exceeds a length of 260 characters an error will occur. This is particularly unfortunate when copying or moving many files at once and when the process stops before completion. As the length of a file path mainly depends on the lengths of its components, we suggest to restrict

- folder names to no more than 20 characters and
- file names to no more than 50 characters.

This would allow a file path to contain nine subfolder names at maximum. The maximum number of subfolders, i.e. the maximum folder depth, should be kept small by following best-practices in Folder Structures. If folder or file names are generated by software (e.g. logger software, modelling software, or reference manager) please check if the software allows to modify the naming scheme. If we nevertheless have to deal with deeply nested folder structures and/or very long file or folder names we should store them in a flat folder hierarchy, i.e. not in

```
\\server\projekte$\department-name\projects\project-name\
  data-work-packages\work-package-one-meaning-the-following\modelling\
  scenario-one-meaning-the-following\results.
```

Rule E: Formatting of Dates and Numbers

When adding date information to file names, please use one of these formats:

- `yyyy-mm-dd` (e.g. `2018-06-28`)
- `yyyymmdd` (e.g. `20180628`)

By doing so, file or folder names that differ only in the date will be displayed in chronological order. Using the first form improves the visual distinction of the year, month and day part of the date. Using hyphen instead of underscore will keep these parts together when splitting the name at underscore (see above).

When using numbers in file or folder names to bring them into a certain order, use leading zeros as required to make all numbers used in one folder level have the same length. Otherwise they will not be displayed in chronological order in your file browser.

Example:

- `01`, `02`, `03`, etc. if there are 9 to 99 files/folders or

- `001`, `002`, `003`, etc. if there are 100 to 999 files/folders.

Rule F: Allowed Words

We recommend to define sets of allowed words in so called vocabularies. Only words from the vocabularies are then expected to appear as words in file or folder names. Getting accustomed to the words from the vocabularies and their meanings allows for more precise file searching. This is most important to clearly indicate that a file or folder relates to "special objects", such as projects or organisations or monitoring sites. At least for projects and organisations we want to define vocabularies in which "official" acronyms are defined for all projects and all organisations from which we expect to receive data (see the chapter on acronyms). Always using the acronyms defined in the vocabularies allows to search for files or folders belonging to one specific project or being provided by one specific organisation.

We could also define vocabularies of words describing other properties of a file or folder. We could e.g. decide to always use `clean-data` instead of `data-clean`, `cleaned-data`, `data-cleaning`, `Datenbereinigung`, `bereinigte-daten`, and so on.

Rule G: Order of Words

We could go one step further and define the order in which we expect the words to appear in a file or folder name. Which types of information should go first in the filename? The order of words determines in which way files are grouped visually when being listed by their name. If the acronym of the organisation goes first, files are grouped by organisation. If the acronym of the monitoring site goes first, files are grouped by monitoring site. According rules cannot be set on a global level, i.e. for the whole company or even for a whole project. The requirements will be different depending on the type of information that are to be stored. We recommend to define naming conventions where appropriate and to describe them in a metadata file in the folder below which to apply the naming convention.

Rule H: Allowed Languages

Do not mix words from different languages within one and the same file or folder name. For example, use `regen-ereignis` or `rain-event` instead of `regen-event` or `rain-ereignis`.

Within one project, use either only English words or only German words in file or folder names. This restriction may be too strict. However, I think that we should follow this rule at least for the top level folder structures. It is not nice

that we see folders `AUFTRAEGE` (German) and `GROUNDWATER` (English) as folder names within the same parent folder.

2.2.3 Folder Structures

"Here are some tips from digital asset management expert Edward Smith:

- Create a template: Copy and paste it every time you start a new project or task. Or, even better: save yourself from the hassle of manually re-creating your structure over and over again bysetting up a Zap to do it for you.
- Think of folder names as keywords: Keep in mind that you can search for files using folder names; the more specific, the more quickly you'll find what you're looking for.
- Keep folders unique: Make sure there's no overlap in what goes into your folders (e.g., there shouldn't be two places you're keeping invoices for the same project).
- Make a cheat sheet: It's OK if you don't have every single folder memorized. There's no shame in saving a flow chart for quick reference." (https://zapier.com/blog/organize-files-folders/)

Restrictions/Conventions:

- Each top-level folder should represent a project, i.e. should be defined in the top level file PROJECTS.txt.
- Each possible owner should be defined in the top level file ORGANISATIONS.txt.
- The naming convention for the organisations is the same as for projects.

Metadata About Folder Structures

We recommend to describe the meaning of subfolders in a file README.yaml in the folder that contains the subfolders.

Example for such a README file:

```
rlib:
  created-by: Hauke Sonnenberg
  created-on: 2019-04-05
  description: >
    R library for packages needed for R training at BWB.
    To use the packages from that folder, use
    .libPaths(c(.libPaths(), "C:/_UserProgData/rlib"))
```

```
rlib_downloads:
  created-by: Hauke Sonnenberg
  created-on: 2019-04-05
  description: >
    files downloaded by install.packages(). Each file represents a package
    that is installed in the rlib folder.
```

Restrictions/Conventions:

- Each top-level folder should represent a project, i.e. should be defined in the top level file `PROJECTS.txt`.
- Each possible owner should be defined in the top level file `ORGANISATIONS.txt`.
- The naming convention for the organisations is the same as for projects.

s)

Project Folder Structure

We said that we want to concentrate on the folder structures within the project folders. Nevertheless, we would like to give a recommendation on how the project folders could be organised within its top level folder. In this structure, there are no subfolders for the different departments any more.

```
//server/projects$
  PROJECTS.txt
  project-1/
  project-2/
  project-3/
```

In the `projects$` folder

- each subfolder name should appear in the file `PROJECTS.txt`
- there should not be any folder on the top level that does not represent a project.
- there should be no other files on the top level as the files that are described in this documentation.
- there are no subfolders representing departments any more. The mapping of projects to departments is done in the file `PROJECTS.txt`

Rawdata Folder Structure

We propose to organise raw data by project first and by origin (i.e. source or owner) of the data second. We will create a network folder //server/rawdata$ in which all files have set the read-only property. We suggest to store raw data

by project first and by the organisation that owns (i.e. generated, provided) the data second. This could look like this:

```
//server/rawdata

  ORGANISATIONS.txt

  PROJECTS.lnk [Symbolic Link to PROJECTS.txt in //server/projects$]

  test-project/
    bwb/
      rain/
        METADA/
        rain.xls
      laboratory/
        METADATA/
        laboratory.xls

    kwb/
      discharge/
        METADATA/
        q01.csv
        q02.csv
        q03.csv
```

Data Processing Folder Structure

In the data processing area the files are stored by project first. Within each project data may be organised by topic and/or data processing step.

```
//server/processing

test-project/
  01_data-cleaning
    METADATEN
    rain_raw.csv
    rain.csv
    quality.csv
    discharge.csv
  02_modelling
    summer
    winter
    VERSIONEN
      v0.1
      v1.0
        summer
```

```
      winter
software
```

Results folder structure

The structure in the result data area should represent the project structure. It could e.g. be organised by working package. When being organised by working package the folder names should not only contain the working package number but also indicate the name of the working package.

```
//server/projects

  test-project/
    Data-Work Packages
      wp-1_monitoring
      wp-2_modelling
        summer.lnk # symbolic links to last version
        winter.lnk # in data processing
```

2.2.4 Versioning

Versioning or version control is the way by which different versions and drafts of a document (or file or record or dataset or software code) are managed. Versioning involves the process of naming and distinguishing between a series of draft documents that leads to a final or approved version in the end. Versioning "freezes" certain development steps and allows you to disclose an audit trail for the revision and update of drafts and final versions. It is essential to reproduce results that may be based on older data.

Manual

Manual versioning may costs more time and requires some discipline, but ensures long-term clean and generally understandable file structure and provides a quick overview of the actual status of development. Manual versioning does not require additional software (except a simple text editor) and is realized by following these simple guidelines:

- A version is created by copying the current file and pasting it to a subfolder named VERSIONS
- Each successive draft of a file in the VERSIONS folder is numbered sequentially from e.g. `v0.1`, `v0.2`, `v0.3` as a postfix at the end of the file name (e.g. `filename_v0.1`, `…v0.2`, `…v0.3`, and so on)

- Finalised forms (e.g. the presentations was held on a conference, the report was reviewed) become entitled with a new version number, e.g. `v1.0`, `v2.0` and so on
- Read-only is applied to each versioned file (to prevent accidental loss of final versions of files)
- Only files without version name as postfix are modified
- A `VERSIONS.txt` is created and kept up-to-date with a text editor, containing meta information on purpose of the modification and the person who made it

It is noteworthy that "final" does not necessarily mean ultimately. Final forms are subject to modification and it is sometimes questionable whether a "final" status has been reached. Therefore, it is more important to be able to track the modifications in the `VERSIONS.txt` rather than arguing on version numbers.

Example:

```
BestPractices_Workshop.ppt
VERSIONS/
  + VERSIONS.txt
  + BestPractices_Workshop_v0.1.ppt
  + BestPractices_Workshop_v0.2.ppt
  + BestPractices_Workshop_v1.0.ppt
```

Content of file `VERSIONS.txt`:

```
BestPractices_Workshop.ppt
- v1.0: first final version, after review by NAME
- v0.2: after additions by NAME
- v0.1: first draft version, written by NAME
```

Automatic

Automatic versioning is mandatory in case of programming.

The versioning is done automatically in case a version control software, like Git or Subversion are used.

At KWB we currently use the following version control software:

- Subversion: for internally storing programm code (e.g. R-scripts/packages) we have an Subversion server, which is accessible from the KWB intranet. However, this requires:
  - the installation of the client software TortoiseSVN and a
  - valid user account (for accessing the server) which is currently provided by the IT department on request

- Git: for publishing programm code (e.g. R packages) external on our KWB organisation group on Github. Currently all repositories are public (i.e. are visible for everyone), but also use of private repositories is possible for free as KWB is recognised as non-for-profit company by Github, offering additional benefits for free

Use of version control software is required in case of programming (e.g. in R, Python, and so on) and can be useful in case of tracking changes in small text files (e.g. configuration files that run a specific R script with different parameters for scenario analysis).

Drawbacks:

- Special software (TortoiseSVN), login data for each user on KWB-Server and some basic training are required
- In case of collaborate coding: sticking to 'best-practices' for using version control is mandatory, e.g.:
  - timely check in of code changes to the central server,
  - Speaking to each other: so that not two people work at the same time at the same program code in one script as this leads to conflicts that need to be resolved manually, which can be quite time demanding. You are much better off if you avoid this in the upfront by talking to each other

Advantages:

- Only one filename per script (file history and code changes are managed either internally on a KWB server in case of using TortoiseSVN or externally for code hosted on Github)
- Old versions of scripts can be restored easily
- Additional comments during `commit` (i.e. at the time of transfering the code from the local computer to the central version control system about `why` code changes were made and build-in diff-tools for tracking changes improve the reproducibility

Attention: version control software is not designed for versioning of raw data and thus should not be used for it. General thoughts on the topic of 'data versioning' are available here: https://github.com/leeper/data-versioning

A presentation with different tools for version control is available here: https://www.fosteropenscience.eu/node/597

2.2.5 Organising E-Mails

> "Most people now routinely send and receive lots of messages every day and as a Yesult, their inbox can get very quickly overloaded with hundreds of personal and work-related email. Setting aside some time to organise your emails will ensure information can be found quickly and easily, and is stored securely. Why should I organise my email? Apart from the obvious frustration and time wasted looking for that email you remember sending to someone last month, email is increasingly used to store important documents and data, often with information related to the attachments within the email itself."
>
> — University of Cambridge (2018)

How can I ensure my emails remain organised?

Here are some general tips to ensure your email remains organised in the long term University of Cambridge, 2018:

- Delete emails you do not need. Remove any trivial or old messages from your inbox and sent items on a regular (ideally daily) basis.
- Use folders to store messages. Establish a structured file directory by subject, activity or project.
- Separate personal emails. Set up a separate folder for these. Ideally, you should not receive any personal emails to your work email account.
- Limit the use of attachments. Use alternative and more secure methods to exchange data where possible (see 'data sharing' for options). If attachments are used, exercise version control and save important attachments to other places, such as a network drive.

2.2.6 Managing References

For managing reference there are plenty of tools available. A detailed overview is provided by (Fenner et al., 2014) in the Chapter Reference Management At KWB we use Endnote, for which an internal guideline document (KWB-EndNote-Guideline-v02.pdf,) was developed.

2.2.7 Data Preservation

Data must be retained to support your research findings

Standards: ?5 years?

2.3 Metadata

Metadata are data about data. It is up to us to define

- what metadata (about raw data, processed data, produced plots, documents, reports, etc.) we want to store,
- where to store metadata and
- in what format to store metadata.

We plan to specify the requirements in more detail when dealing with the test projects (case studies). Then, will also check metadata standards.

Metadata about raw data should always be stored.

Metadata about processed data are also important. However, in case of automated processing with a script, it may be possible to deduce the content of a generated file from the content of the script.

2.3.1 Why Metadata?

Why should we store metadata about data? Metadata are required to

- interpret raw data, e.g. "Why are the oxygen values so high? Ah, I see, someone was there to clean the sensor!",
- gain an overview about available data,
- know what we are allowed to do with the data.

2.3.2 What Metadata to Store?

General

What information would someone need to find/re-use your data? E.g.

- Location,
- Title,
- Creator name,
- Description,
- Date collected?

Metadata about Raw Data

What are the most important information about raw data that we receive?

- Obtained from whom, when, via whom and which medium, e.g.
  - `E-Mail from A to B on 2018-01-25` or
  - `USB-Stick given personally from C to D on 2018-01-26`
- Restriction of usage, e.g.
  - `only for project x` or
  - `only within KWB` or
  - `must not be published!` or
  - `should be published!`!
- Description of content and format
  - Where measurements were taken?
  - What methods were used (to take samples, to analyse parameters in the laboratory)?
  - What devices where used?
  - What do the columns of the table (in the database, XLS/CSV file) mean?
  - In what units are the values given?

Metadata about Processed Data

What are the most important information about the data that we produce?

- Who created the file? If the file was created by a script, what script created the file and who ran the script?
- When was the file created?
- What was the input data (e.g. raw data or preprocessed data)?
- Which methods were applied to generate the output from the input?
- What was the environment, what were boundary conditions, e.g.
  - versions of software,
  - versions of R packages?

Metadata about Programming Scripts

- What does the script do?
- Who wrote the script?
- How to use the script? Give a short tutorial.

Regarding R programming, we should consider providing scripts in the form of R packages. The R packaging system defines a framework of how to answer all of the above questions. See how we did this (not yet always with with a tutorial) with our packages on GitHub.

2.3.3 Where to Store Metadata?

The two main options of storing metadata are:

1. together with the data or near to the data i.e. in the same folder in which the data file resides,
2. in a central file or database.

Unless we have a professional solution (i.e. software on metadata management) we should prefer the first approach which is simpler and more flexible than the second one.

2.3.4 In What Format to Store Metadata?

Metadata should be stored in a simple plain (i.e. not formatted) text format. This format can be read and written by any text editor on different operating systems and does not require any specific software. And most important: it is human readable.

In the simplest form, metadata can be stored in a plain text file `README.txt`. As stated above, this file should reside in the same folder as the files that contain the data to be described.

Please keep in mind: It is better to write something than to write nothing. It does not have to be perfect. Just try the best yout can at the moment that you are storing the data. Try to write the metadata directly after storing the data, do not wait until you "feel in the mood" to do so. Anyone who is later working or planning to work with the data may be grateful finding some information on it.

Better than writing an unstructured text file is to write a structured text file. The so called YAML format is such a structured format. We want to use this standard. It seems to have established in the scientific world. The advantage of a structured format is that reading of this format can be automated. We aim

at collecting all available metadata by automatically browsing for YAML files, reading their content and creating overviews on available files and data.

Once you have written a YAML file you can check the validity of the format with this online validator: https://ci.appveyor.com/tools/validate-yaml

2.3.5 Metadata Management Tools

Tools for metadata tracking and data standards are:

- metadata editor, e.g. online editor of GFZ Potsdam

2.3.6 Metadata Standards

We want to use a metadata standard.

Examples for metadata standards are (in brackets are listed the institutions who publish their data by using this standard):

- DataCite metadata scheme
  - Leibniz-Zentrum für Agrarlandschaftsforschung e. V. (ZALF)
  - Deutsches GeoForschungsZentrum Potsdam (GFZ) (+ ISO + DIF + Dublin Core)
- ISO 19115-2
  - Bundesanstalt für Gewässerkunde (BfG)
  - Alfred Wegener Institute, Helmholtz Centre for Polar and Marine Research (AWI)

Best-practices roadmap:

1. Check meta data standards, e. g. DataCite (see also: ZALF, GFZ Potsdam)
2. Define minimum metadata requirements at KWB for raw and processed data.
   projects.

The 'best-practices for metadata' will be developed for the case studies which are assessed within FAKIN project.

We propose to define some special files that contain metadata related to files and folders. To indicate that these files have a special meaning, the file names are all uppercase.

2.3.7 Special Metadata Files

As stated earlier, we want to use consistent, unique identifiers to indicate the belonging of data to a certain project or data owner. We propose to define the identifiers in terms of special metadata files.

2.3.7.1 Metadata File PROJECTS.txt and related files

The project identifiers are defined in a simple YAML file `PROJECTS.yml` in the `//server/projects$` folder. Only the identifiers defined in this file are expected to appear as top-level folder names in the project folder structure within this network drive.

Possible content of `PROJECTS.yml`:

```
flusshygiene:
  department: suw
  short-name: Flusshygiene
  long-name: >
    Hygienically relevant microorganisms and pathogens in multifunctional water
    bodies and hydrologic circles - Sustainable management of different types of
    water bodies in Germany
  financing:
    funder: bmbf
    sponsor: bwb
ogre:
  department: suw
  short-name: OGRE
  long-name: >
    Relevance of trace organic substances in stormwater runoff of Berlin
  financing:
    funder: uep-2
    sponsor: Veolia
optiwells-2:
  department: grw
  short-name: OPTIWELLS 2
  long-name:
  type: sponsored
reliable-sewer:
  department: suw
  short-name: RELIABLE_SEWER
  long-name:
  type: contracted
smartplant:
  department: wwt
  short-name: Smartplant
```

```
  type: sponsored
...
```

In the file `PROJECTS.yml` each entry represents a project. Each project is identified by its identifier. The project acronyms appear in alphabetical order. Each entry should at least contain information on the department, a short/long name or title of the project and the type of project (funded, sponsored, contracted). Additional information such as the year of the start of the project could be given.

It could also be useful to define three letter codes for projects. These codes could e.g. be used in tables or diagrams in which different projects are compared and in which space may be limited.

The department acronyms could be defined in a file `DEPARTMENTS.yml`:

```
grw:
  short-name: Groundwater
  head: Hella Schwarzmüller
suw:
  short-name: Urban Systems
  head: Pascale Rouault
wwt:
  short-name: Process Innovation
  head: Ulf Miehe
```

The funder acronyms could be defined in a file `FUNDERS.yml`:

```
bmbf: German Federal Ministry of Education and Research (BMBF)
uep-2: >
  Umweltentlastungsprogramm des Landes Berlin, co-financed by the European Union
  (UEP II)
```

2.3.7.2 Metadata File ORGANISATIONS.txt

It is very important to know the origin or owner of data. This is an important piece of metadata. Therefore we define unique identifiers for the owners of data that we use. The acronyms are defined in a special file `ORGANISATIONS.txt`. Possible content of this file:

```
bwb: Berliner Wasserbetriebe
kwb: Kompetenzzentrum Wasser Berlin
uba: Umweltbundesamt
```

2.4 Data Collection

Data are often inconsistent, incomplete, incorrect, or misspelled. Data cleaning is essential.

For data cleaning you may use a GUI (Graphical User Interface) based tool like OpenRefine http://openrefine.org/ or choose a programmatic approach.

In the following we describe how data can be imported into the R-Programming Environment, which can be used for data cleaning, aggregation and visualisation. (Grolemund and Wickham, 2017)

2.4.1 Logger Devices

The R package kwb.logger (Sonnenberg, 2018) helps to import raw data from loggers used in different KWB projects into the software R (R Core Team, 2017), which is used for data processing (e.g. data cleaning, aggregation and visualisation).

For details, which loggers currently are supported by the R packages kwb.logger please check the documentation website.

2.4.2 Spreadsheets

General recommendations for working with EXCEL spreadsheets is given in the FAQs section.

2.4.2.1 Import Data From One Excel File

- Save the original file in the rawdata zone.

2.4.2.2 Import Data From Many Excel Files

2.4.2.2.1 Files Are In the Same Format

Import Excel files of the same format by

- defining a function that is able to read the data from that file
- calling this function in a loop for each file to import.

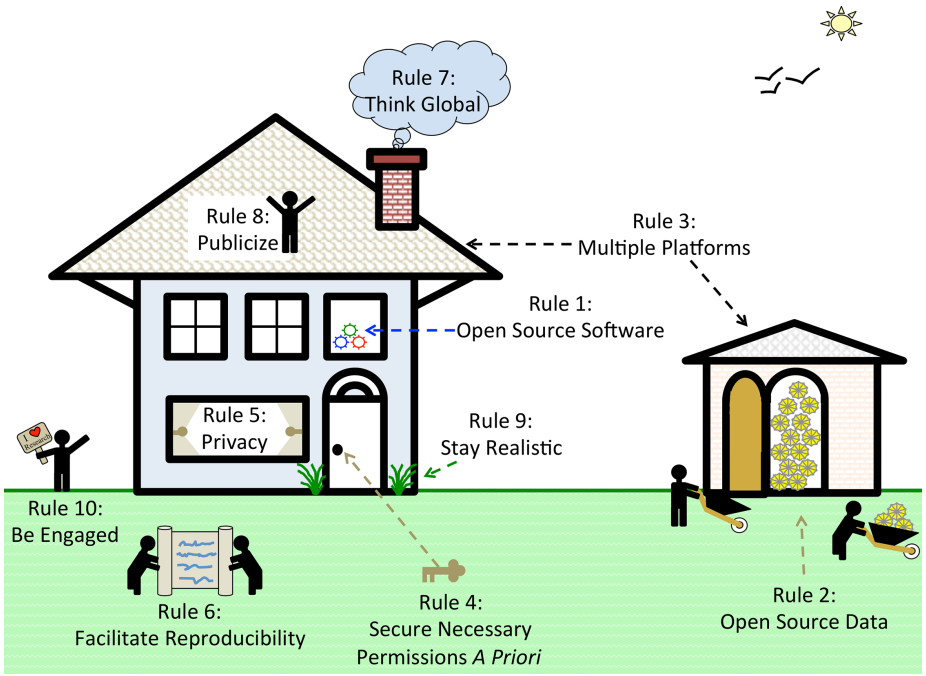


Figure 2.1: Modern life context for the ten simple rules (Boland et al., 2017)

2.4.2.2.2 Files Are In Different Formats

We developed a general approach of importing data from many Excel files in which the formats (e.g. more than one table area within one sheet, differing numbers of header rows) differ from file to file.

2.5 Data Publishing and Sharing

> "This figure provides a framework for understanding how the "Ten Simple Rules to Enable Multi-site Collaborations through Data Sharing" (Boland et al., 2017) can be translated into easily understood modern life concepts.
>
> **Rule 1** is Open-Source Software. The openness is signified by a window to a room filled with algorithms that are represented by gears.
>
> **Rule 2** involves making the source data available whenever possible. Source data can be very useful for researchers. However, data are

often housed in institutions and are not publicly accessible. These files are often stored externally; therefore, we depict this as a shed or storehouse of data, which, if possible, should be provided to research collaborators.

Rule 3 is to "use multiple platforms to share research products." This increases thechances that other researchers will find and be able to utilize your research product—this is represented by multiple locations (i.e., shed and house).

Rule 4 involves the need to secure all necessary permissions a priori. Many datasets have data use agreements that restrict usage. These restrictions can sometimes prevent researchers from performing certain types of analyses or publishing in certain journals (e.g., journals that require all data to be openly accessible); therefore, we represent this rule as a key that can lock or unlock the door of your research.

Rule 5 discusses the privacy issues that surround source data. Researchers need to understand what they can and cannot do (i.e., the privacy rules) with their data. Privacy often requires allowing certain users to have access to sections of data while restricting access to other sections of data. Researchers need to understand what can and cannot be revealed about their data (i.e., when to open and close the curtains).

Rule 6 is to facilitate reproducibility whenever possible. Since communication is the forte of reproducibility, we depicted it as two researchers sharing a giant scroll, because data documentation is required and is often substantial.

Rule 7 is to "think global." We conceptualize this as a cloud. This cloud allows the research property (i.e., the house and shed) to be accessed across large distances.

Rule 8 is to publicize your work. Think of it as "shouting from the rooftops." Publicizing is critical for enabling other researchers to access your research product.

Rule 9 is to "stay realistic." It is important for researchers to "stay grounded" and resist the urge to overstate the claims made by their research.

Rule 10 is to be engaged, and this is depicted as a person waving an "I heart research" sign. It is vitally important to stay engaged and enthusiastic about one's research. This enables you to draw others to care about your research."

—- (Boland et al., 2017)

Recommended literature:

- Ten simple rules to enable multi-site collaborations through data sharing (Boland et al., 2017)
- Guidelines for publishing (PhD) research data (Kaden and Kleineberg, 2018)

2.5.1 Repositories

Repositories for permanently deposing data are for example:

- General
  - Figshare,
  - Zenodo (a joint project between OpenAIRE and CERN),
  - Mendeley data,
  - Dataverse,
  - Dryad
- Focus on environmental and earth sciences
  - Pangea
  - GFZ Potsdam data services

Repositories for publishing program code are:

- Github or
- Gitlab.

However, both do not offer long term data preservation by default, but using Github it is posible to make the code citable by linking it with Zenodo (see: https://guides.github.com/activities/citable-code/).

We are currently using the following three repositories for publishing program code (mainly R packages):

- Github: for developing and publishing program code (mainly R packages) we use https://github.com/kwb-r. Currently 81 (i.e. 38 public and 43 private) repositories are published on this Github account. For all 32 public R packages there is also a detailed status report available available at https://kwb-r.github.io/status/ , e.g. with information on license, documentation and the "health" of the R package (i.e. whether it can be successfully installed on Linux or Windows platforms).
- Zenodo: for automatically getting a DOI for each software release made in one of our public Github repositories, e.g. aquanes.report (for details see: https://guides.github.com/activities/citable-code/) and

- Gitlab: as backup mirror (https://gitlab.com/kwb-r) for all of our currently 81 (i.e. 34 public and 47 private) repositories currently published on our Github account (https://github.com/kwb-r)

Proposal: define company-wide QMS policy ("top-down") for publishing program code

The above workflow was established from "bottom-up" (i.e. Michael Rustler and Hauke Sonnenberg) with the idea in mind to make the code as open as possible (e.g. by chossing the permisse MIT license as default for all of our public R packages).

However, up to now there is no company wide strategy ("top-down") defined yet that would legitimate this "bottom-up" approach. This creates uncertainty (e.g. what can be published?), so that much more code than necessary is labelled as "private". To reduce this uncertainty the following QMS policy is proposed, which should be discussed and agreed on in one of the next KWB management meetings:

- **Sponsor projects (e.g. funded by BMBF, EU)**: source code will be published by default at https://github.com/kwb-r in **public** repositories (i.e. it will be accessible for everyone) under the permissive MIT license in case that the source code does not:
  - contain security critical paths (e.g. to our company server) or
  - confidential data.

  Code should be developed in such a way that both of the criteria (security critical paths, confidential data) defined above are considered. Making the code openly available will decrease the burden to install them (e.g. not each student needs to get an "access" token to install private repositories, as required for "contract" projects, see below).

- **Contract projects (e.g. funded by BWB, Veolia)**: will be published in **private** repositories by default at https://github.com/kwb-r in case the funder does not pre-define a specific repository. Access to the source code is thus resticted to KWB researchers and students working in the contract project. Project partners and funders can access the source code only if they get an "access token" from the KWB project team.

A blog post by Bosman and Kramer (2016) provide results of a large survey carried out in 2015 among more than 15000 researchers. Insights can be gained on:

- Which scholary communications tools are used and
- Are there disciplinary differences in usage?

> They finally summarise: "Another surprising finding is the overall low use of Zenodo – a CERN-hosted repository that is the recommended archiving and sharing solution for data from EU-projects and -institutions. The fact that Zenodo is a data-sharing platform that is available to anyone (thus not just for EU project data) might not be widely known yet."

2.5.2 ORCID

Problem:

> "Two large challenges that researchers face today are discovery and evaluation. We are overwhelmed by the volume of new research works, and traditional discovery tools are no longer sufficient. We are spending considerable amounts of time optimizing the impact—and discoverability—of our research work so as to support grant applications and promotions, and the traditional measures for this are not enough. — (Fenner and Haak, 2014)

Solution:

> "Open Researcher & Contributor ID (ORCID) is an international, interdisciplinary, open and not-for-profit organization created to solve the researcher name ambiguity problem for the benefit of all stakeholders. ORCIDwas built with the goal of becoming the universally accepted unique identifier for researchers:
>
> 1. ORCID is a community-driven organization
> 2. ORCID is not limited by discipline, institution, or geography
> 3. ORCID is an inclusive and transparently governed not-for profit organization
> 4. ORCID data and source code are available under recognized open licenses
> 5. the ORCID iD is part of institutional, publisher, and funding agency infrastructures.
>
> Furthermore, ORCID recognizes that existing researcher and identifier schemes serve specific communities, and is working to link with, rather than replace, existing infrastructures."
>
> — (Fenner and Haak, 2014)

2.5.3 Licenses

> "In most countries in the world, creative work is protected by copyright laws. International conventions, and primarily the Berne Convention of 1886, protect the copyright of creators even across international borders for 50 years after the death of the creator. This means that copying and using the creative work is limited by conditions set by the creator, or another copyright holder. For example, in many cases musical recordings may not be copied and further distributed without the permission of the musician, or of the production company that has acquired the copyright from the musician. Facts about the universe that are discovered through research are not subject to copyright, but the collection, aggregation, analysis and interpretation of research data may be considered creative work, and could be protected by copyright laws. Thus, the consumption of research publications is governed by copyright law. Furthermore, even data sharing is often governed by copyright laws, because the compilation of data to be shared often requires a creative effort. Another case of resarch-relevant copyrighted products is software that is developed in the course of research. In all of these cases, if license terms are not explicitly specified, the work is considered to be protected as"all rights reserved". This means that no one but the creator of the work can use the work unencumbered. For software this means that copying and further distribution of the software is prohibited. Even running the software may be restricted. The exact selection of a license is beyond the scope of this section, but depends on your intentions and goals with regard to the software"
>
> — (Rokem and Chirigati, 2018)

Recommended literature:

- Intellectual Property and Computational Science (Stodden, 2014)
- forschungslizenzen.de (For, 2019)
- Creative Commons Licences (Friesike, 2014)
- choosealicense.com/

2.5.4 File Formats

> "Scientific data is saved in a myriad of file formats. A typical file format might include a file header, describing the layout of the data on disk, metadata associated with the data, and the data itself, often stored in binary format. In some cases (e.g., CSV (or comma-separated value) files), data will be stored as text. The danger of proliferation of file formats in scientific data lies in the need to build

Table 2.1: Suitability of file formats for long-term preservation (Kaden and Kleineberg, 2018)

	More than ten years	Up to ten years	Not suitable
Text	PDF/A, TXT, ASC, XML	PDF, RTF, HTML, DOCX, PPTX, ODT, LATEX	DOC, PPT
Data	CSV	XLSX, ODS	XLS
Pictures	TIFF, PNG, JPG 2000, SVG	GIF, BMP, JPEG	INDD, EPS
Audio	WAV	MP3, MP4	
Video	Motion JPG 2000, MOV	MP4	WMV

> and maintain separate software tools to read, write and process all these data formats. This makes interoperability between different practitioners more difficult, and limits the value of data sharing, because access to the data in the files remains limited."
>
> — (Rokem and Chirigati, 2018)

2.5.5 Data Exchange Standards

WaterML2:

> "...is a new data exchange standard in Hydrology which can basically be used to exchange many kinds of hydro-meteorological observations and measurements. WaterML2 has been initiated and designed over a period of several years by a group of major national and international organizations from public and private sector, such as CSIRO, CUAHSI, USGS, BOM, NOAA, KISTERS and others. WaterML2 has been developed within the OGC Hydrology Domain Working group which has a mandate by the WMO, too."
>
> — WaterML2

ODM2: is an information model and supporting software ecosystem for feature-based earth observations

Chapter 3

Case Studies

The three following case-studies are tested in detail within FAKIN (i.e. proposed best-practices will be applied for this case studies and cross-checked whether their application is useful.

3.1 Geogenic Salination

Adapt and test with new folder drive workflow as proposed

- `rawdata`
- `processing`
- `processing`

3.2 LCA Modelling

Challenge:

The LCA modelling software Umberto can produce large raw data output files (> 300 MB csv files) that sometimes are even to big for EXCEL 2010 (> 1 millions) but need to be aggregated (e.g. grouped by specific criteria). This was usually performed manually within EXCEL in case that model output data was below EXCEL‘s 1 million row limit.

Workflow improvement developed within FAKIN:

An open source R package kwb.umberto was programmed for automating:

- data import the Umberto model results,

- performing data aggregating to the user needs and finally
- exporting the aggregated results in an `results.xlsx` EXCEL spreadsheet.

This `results.xlsx` EXCEL spreadsheet is referenced by another EXCEL spreatsheet `figues.xlsx` (which contains the figure templates and just links to the `results.xlsx` in order to update the predefined figures).

This workflow now reduces the time consuming and error-prone formerly manually performed data aggregation in EXCEL, whilst still enabling the users to adapt the figures to their needs without coding knowledge.

3.3 Pilot Plants

Challenges:

The output of (on-line) monitoring technologies is often difficult to interpret and also inconvenient to handle as the output formats of different devices (in one water treatment scheme) can vary strongly. Furthermore, frequent reporting and documentation of the treatment performance via (on-line) monitoring can be time consuming for the personnel and requires advanced software solutions. An alternative to commercial (and often expensive) software solutions are tools which are based on the open software R (R Core Team, 2017). The free software approach allows any R programmer to produce customized tools for each individual end-user.

Thus an automated reporting tool is developed within the AQUANES project for enabling an integrative assessment of the different monitoring devices and integration with water quality data obtained from analysis in the laboratory for four different pilot plant sites in order to:

- Increase the reliability and reproducibility of handling large amounts of data by reducing the likeliness in human error in complex systems and by increasing the transparency of the data processing.
- Promote the use of customized R tools for different end-user such as utilities, consultants and other research teams.

Therefore the open source R package aquanes.report (Rustler, 2018) was programmed, which is able to:

- import operational and lab data for each pilot site,
- performs temporal aggregation (e.g. 5 min, 1 h, 24h median values),
- visualises raw or aggregated data either interactively in a web browser or by
- creating a standardised report (e.g. monthly) in html, pdf or docx

For the four different pilot plant sites the data (operational and lab data) for being imported into the R tool came from various sources at different temporal resolutions, which are detailed below:

- Haridwar: operational data stored by Autarcon in mySQL database (temporal resolution: ~ 2-3 minutes, i.e. ~ 0.7 million data points per month), which is accessible from the web and thus could be easily imported into R. Lab data was provided by Autarcon initially in a unstructured format, which was impossible to be automatically imported into R. However, after agreeing on a standardised EXCEL spreadsheet format (e.g. one spreadsheet per site, one sheet per parameter and additional sheets providing metadata for parameters and sites) it was possible to integrate the lab data into the R tool.
- Basel Lange-Erlen: operational data is provided by the water supplier in EXCEL spreadsheets on a weekly basis for each site (i.e. "Rein" and "Wiese") with a temporal resolution of 5 minutes (i.e. ~ 0.5 million data points per month). Lab data are provided by the water supplier in a single comma separated csv file, which is exported from a database. Thus the structure of the lab data was standardised and could be easily imported into the R tool.
- Berlin-Schönerlinde: operational data from the WEDECO pilot plant are collected using a SCADA system (~ temporal resolution: seconds, i.e. ~ 10 million data points per month). Lab data are provided by BWB in a single EXCEL spreadsheet. However, its structure often changes in case it is updated by BWB, making an automated importing using R impossible without adapting the import functions. Thus lab data were not integrated in the R tool for this site.
- Berlin-Tiefwerder: operational data from the PENTAIR pilot plant are collected using a SCADA system (~ temporal resolution: ~ seconds, i.e. ~ 10 million data points per month). Lab data are provided by BWB in a single EXCEL spreadsheet. However, its structure often changes in case it is updated by BWB, making an automated importing using R impossible without adapting the import functions. Thus lab data were not integrated in the R tool for this site.

The high temporal high resolution (~ seconds) of the operational data for both Berlin pilot plants resulted in large data amounts (~ 10 million data points per month), which required an large effort to optimise the performance of the R reporting tool in oorder to enable the visualisation of the pilot plant‘s raw data for its test operation period (~ 18 months) on computers with limited RAM ressources (~ 8-12 GB).

The R tool is used by KWB (for the sites Berlin-Schönerlinde and Berlin-Tiefwerder) regulary for assessing the pilot plants‘ operational performance in-

teractively. In addition for an advanced assessment only the data importing and aggregation routines and combined with R scripts developed by KWB students.

For the other two pilot plant sites Haridwar and Basel Lange-Erlen the AQUANES project partners use the automated R reporting tool in a similar way.

Chapter 4

Other Projects

Here we are just suming up the data workflows and created tools in (old) KWB projects, but – in contrast to the case studies – the workflows and tools were not tested explicitly within the FAKIN project.

The goal of this chapter is to enlarge the internal knowledge base at KWB about what and how data challenges have already been successfully solved in the past.

4.1 Spree2011 (2007)

Used data by source (data formats in parentheses)

- KWB:
  - water level and discharge at one monitoring site (Text/CSV)
  - rain (Text/CSV)
- BWB:
  - pumping rates in the pumping stations (Excel)
  - water levels in the pumping stations (Excel)
  - rain at some gauges near the monitoring site (Excel)

Tasks and methods by topic

- Dry-weather and wet-weather calibration of a sewer network model (Infoworks)
  - InfoWorks: Creating rain input files
  - InfoWorks: Creating RTC input files

Questions that arose:

- Where to store presentations (trainee vs. employee)?
- Where to store the raw data (personal drive of the trainee)?

- How does Infoworks interpret timestamps, how do BWB provide timestamps? -> metadata

4.2 MIACSO (2009)

Monitoring

- sites: one site in the sewer (monitoring container), more sites in the river
- variables: water quantity and quality
- devices: online sensors

Modelling

- Sewerage: Infoworks
- River hydraulics: Hydrax
- River quality: QSim

Data storage

- High amount of data -> extra server: `moby`
- We put some effort in planning good folder structures for the data. Nevertheless the structure at the end of the project is not as clean as it was planned.
- Data that we received from project partners was stored in `Daten/EXTERN`.
- Raw data was stored in a folder `Daten/RAW` which was write-protected and required a special user-account for storing new data.

```
Daten/
  ACCESS/ # MS Access databases, containing raw data
  EXTERN/ # External data (by organisation)
  META/   # MS Access databases, containing metadata
          # (about calibration, maintenane, sites, variables)
  RAW/    # Text files containing raw data, from KWB-own devices only, by site
```

Metadata

- many devices in the container -> meta data about device cleaning and maintenance important -> tool: `META_Maintenance.mdb`

Methods and Tools

- We imported most of the data from text files into MS Access databases in -> tool: `MiaCsoRawImport.mdb`
- We calibrated the sensors offline by using SQL queries to provide calibrated data from raw data -> tool: `MiaCsoMetaCalibControl.mdb`
- We used SQL queries to perform data processing -> tool: `MiaCsoStatAnalysis.mdb`
- Data validation (outlier detection) was done in a two step procedure:
  1. Automatic preselection using MS Access tool `MiaCsoStatAnalysis.mdb`
  2. Manual selection using self-developed graphical tool in Origin

Developed Tools:

- MS Access Applications
  - `MetaMaint.mdb`: Monitoring Metadata Management
  - `MiaCsoRawImport.mdb`: Text File Import to MS Access
  - `MiaCsoStatAnalysis.mdb` (project deliverable): Definition and automatic execution of sequences of SQL queries
- Origin extension to interactively select and store outliers graphically
- R packages
  - `kwb.mia.evalCritO2` (project deliverable): graphical evaluation of critical oxygen conditions in the river
  - `kwb.mia.iw`: Calculation of file sizes of InfoWorks result csv-files exported from InfoWorks.
  - `kwb.miacso`: functions used in MIA-CSO, for example for plotting data availabilities.

4.3 KURAS

Developed Tools:

- Frontend for KURAS Database of Rainwater Management Measures: `KURAS_DB_Acc2003_hs.mdb`
- R package `kwb.kuras`: Interface to KURAS database

4.4 OGRE

- Decision to use CUAHSI Community Observations Data Model (ODM)
- R script to import lab data from Excel to MS Access database implementing ODM

Developed Tools:

- R packages
  - `kwb.ogre`
  - `kwb.ogre.model`
  - `kwb.odm`
  - `kwb.odmx`

4.5 Flusshygiene

- Adaptation of free online monitoring data visualisation HydroServerLite
- Reusage of lab data import script developed in OGRE

4.6 DEMEAU

Developed R packages for groundwater modelling:

- kwb.hantush (Rustler, 2016a)
- kwb.vs2dh
- kwb.demeau

4.7 DEMOWARE

R package kwb.qmra (Rustler, 2016b) is a generic QMRA (Quantitative Microbiological Risk Assessment) calculation engine assessing the performance of water supply systems.

It was successfully applicated for the Old Ford wasterwater treatment plant (Kraus et al., 2016; Rustler, 2016c). A detailed documentation for the R package and its usage is available online.

4.8 OPTIWELLS

Created R packages:

- kwb.wtaq (Sonnenberg and Rustler, 2016): groundwater modelling, e.g. for assessing the impact of production well (e.g. well diameter, pumping rates) and aquifer characteristics (e.g. hydraulic conductivity) on the resulting drawdawn. A detailled tutorial is available online,
- kwb.epanet: wrapper for (pressurised)pipe network simulation model EPANET

4.9 RWE

(Semi)automated creation of a complex groundwater simulation model with MODFLOW-2005 for the project Maxflow. The developed model consisted of up to three model layers with up to 1000 abstraction wells with temporal-spatial varying pumping rates within the simulation period. For automating the model generation the Python package flopy v3.2.5 (Bakker et al., 2016b,a) was used with some minor modifications by KWB (see here). Github was used as version control software for tracking changes in the code and each model scenario was stored in its Github branch (for details, see here).

Chapter 5

Literature Review

5.1 Data Storage

5.1.1 Ten Simple Rules for Digital Data Storage

The following rules are defined by Hart et al. (2016):

> Rule 1: Anticipate How Your Data Will Be Used
>
> Rule 2: Know Your Use Case
>
> Rule 3: Keep Raw Data Raw
>
> Rule 4: Store Data in Open Formats
>
> Rule 5: Data Should Be Structured for Analysis
>
> Rule 6: Data Should Be Uniquely Identifiable
>
> Rule 7: Link Relevant Metadata
>
> Rule 8: Adopt the Proper Privacy Protocols
>
> Rule 9: Have a Systematic Backup Scheme
>
> Rule 10: The Location and Method of Data Storage Depend on How Much Data You Have
>
> — (Hart et al., 2016)

Chapter 6

FAQs

6.1 Naming

Why is the space character not allowed?

Why is the space character not allowed in file or folder names?

File paths containing spaces need to be put into quotes when being used in a command line, such as in:

```
"C:\Programs\model software xyz\run.exe" "\\server\path with space\input.txt"
```

With all paths lacking the space character you can omit the quotes:

```
C:\Programs\model-software-xyz\run.exe \\server\path-without-space\input.txt
```

Consequent usage of paths without spaces allows for simpler programming and leads to less errors during program execution.

Why are special characters not allowed in file or folder names?

The way in which special characters are encoded may differ from system to system, especially between systems with different language settings. File names containing German special characters are e.g. not shown correctly on a French computer and vice versa.

Excluding (e.g. language-specific) special characters from the set of allowed characters in file or folder names avoids problems when exchanging files between partners in different countries.

We may have a problem with a file from Sofia, called

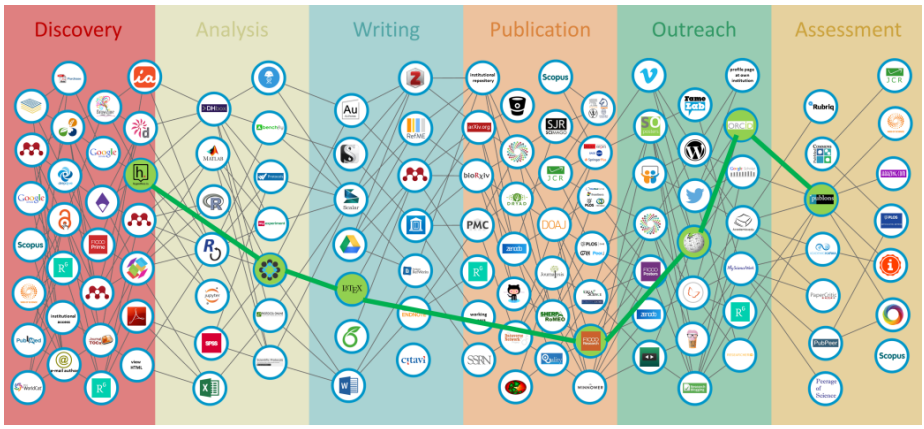
`Example file from Canasoft . , . _Troncons.xml`

and they may have a problem with a file from us, called

`abgebrochene schmuz- und regenwasserhaltungen schützenstraße nord 2. ba.xls`

6.2 Tools for Researchers

A very exhaustive overview of tools used in the researcher's workflow are provided by Bosman and Kramer (2018), which are grouped according to different research phases as shown below.



6.3 Writing More Robust R Code

For details on making R code more robust to work on different computers please read the following tutorial.

Recommended literature:

- Good enough practices in scientific computing (Wilson et al., 2017)
- Ten simple rules for making research software more robust (Taschuk and Wilson, 2017)
- R programming books (freely available online!)
  - R for data science (Grolemund and Wickham, 2017)
  - Advanced R (Wickham, 2015a, 2018)
  - R packages (Wickham, 2015b)

6.4 Learning R on DataCamp

Since June 2017 we have an account for the online learning platform DataCamp. This platform provides many online courses in the programming language R that we use a lot at our institute. The usage of DataCamp allows to let new students and beginners teach themselves in programming. This helps a lot in bringing the programming skills within our institute up to a certain level.

As the institution "KWB" we have a Free group account and a Premium group account. The premium group account comprises ten seats, i.e. ten persons can work with the online tutorials at the same time.

For using DataCamp at KWB the following workflow applies:

1. Become a member in our free R@KWB DataCamp group by clicking on the following link. In case you do not have a DataCamp account yet, please register yourself first by using your KWB email address. The membership in this group serves two purposes: on the one hand you should start on DataCamp with the free courses like for example Introduction to R first. On the other hand this group serves us as a kind of long term memory, to assess who and how frequently the DataCamp courses are used as the amount of users in the free group is – in contrast to our paid premium DataCamp group – not limited.
2. If you want to do a paid DataCamp course, please contact one of our **DataCamp admins** (***see box below***)) who will add you to our corporate premium group in case there is a free seat available. Subsequently you are able to start any of the available premium DataCamp courses.
3. As soon as you either do not want to or you do not have the time for a longer period do one of the paid DataCamp courses, please get in touch with one of our **DataCamp admins** (***see box below***) again, so that he can make your seat available for others at KWB who are interested in learning R.

Admins for our corporate premium group on DataCamp are:

- Nicolas Caradot
- Michael Rustler
- Hauke Sonnenberg

6.5 Using Subversion at KWB

For details on how to use Subversion at KWB please read the following tutorial

6.6 How to Build Your Own Kwb Styled R Package?

For building your own R package from scratch we developed a helper R package kwb.pkgbuild (available on Github), which builds a KWB styled skeleton for your future R package.

With the help of this tool Andreas Matzinger was able to convert this R scripts on resilience within a few hours into the R package kwb.resilience which is now available on Github.

For more details on turning your own code into a R package checkout the tutorial at the package documentation website.

Recommended literature:

- R packages (Wickham, 2015b)
- Advanced R (Wickham, 2015a)

6.7 How to Install KWB R Packages?

Most R packages developed at KWB are not only available in the intranet but are also available on Github. Please check the following website: http://kwb-r.github.io/status/ or https://github.com/KWB-R

Installation of these R packages can be performed with the following command in R(studio):

```
### Required to install an R package from Github
install.packages("devtools", repos = "https://cloud.r-project.org")

### Now install your desired R package (e.g. "kwb.resilience")
devtools::install_github("kwb-r/kwb.resilience")
```

However, in case your required R package that is not yet Github please read the following tutorial

6.8 Working with Excel

Excel often crashes in case:

- a formula is applied for a whole column (i.e. 1 million rows)
- a lot of data is processed

Solution: data in own spreadsheet file

Hauke's "best Practices" for EXCEL (unvalidated, to be discussed!)

- Trennung zwischen Eingabe, Verarbeitung und Ausgabe zumindest auf Tabellenblattebene, d.h. ein Tabellenblattt (oder mehrere) für Eingabe, eines (oder mehrere) für Verarbeitung, eines (oder mehrere) für formatierte Ausgabe und / oder Diagramme
- ggf. Aufteilen auf mehrere Dateien. Das hätte den Nachteil, dass nicht mehr alles in einer Datei ist und nicht so leicht übergeben werden kann. In jedem Fall müsste eine Namenskonvention getroffen werden, z.B.`<file_name>`_input.xlsx, `<file_name>`_calc.xlsx, `<file_name>`_output.xlsx
- Verwenden der relativ neuen EXCEL-Features `Als Tabelle` formatieren.

Vorteil: Formeln können auf ganze Tabellenspalten angewendet werden; Spaltennamen anstatt Zellbezüge mit (unsprechenden) Buchstaben und Zahlen. z.B. Formel für Spalte `Volumen_L`: `= Durchfluss[@[Q_L_s]] * 60 * 5`

- Ein Tabellenblatt pro Tabelle
- Genau eine Headerzeile pro Tabelle mit eindeutigen Spaltennamen
- Ein Tabellenblatt, das die Bedeutung der Spaltennamen erläutert mit Spalten `Tabellenblatt; Spalte; Bedeutung; Einheit; Formel` Vorteil: Dieses Tabellenblatt sollte ausreichen, um die wesentlichen Berechnungen zu verstehen.

Drawback: needs to be permanently kept up-to-date!

- Hilfsspalten mit (dadurch benannten) Zwischenberechnungen anstatt Wiederholung von langen Ausdrücken in Formeln

A general online workshop on the topic Data Organisation in Spreadsheets is provided for free by the DataCarpentry organisation.

copied from `10_old_german_chapters.Rmd` :

Best Practices zum Arbeiten mit Excel:

- z.B. Zellbezüge benennen, dadurch werden Formeln besser lesbar

Ihr werdet es kaum glauben, aber am Anfang meiner Zeit am KWB habe ich noch mit Excel gearbeitet. Ich habe auch komplexe Sachen gemacht und auch Excel-Makros programmiert. Ein Beweis findet sich in meiner persönlichen Logdatei:

> Fr, 14.09.07 08:15-19:30 benutzerdefinierte Excelfunktionen zur Modellgüte in Personl.xls, Modul1; TW-Kalibrierung Wochentag fertig

Ich würde heute nicht mehr empfehlen, Excel-Makros zu programmieren. Es ist umständlich, es gibt keine Bibliotheken und der Quellcode lässt sich nicht unabhängig von der Exceldatei verwalten, so dass keine ordentliche Versionsverwaltung möglich ist. Und wir wollen nicht mehr ohne Versionsverwaltung programmieren!

Wenn wir programmieren, dann sollten wir das einheitlich in R tun. R ist frei, es gibt eine großartige Community und wir haben mittlerweile eine große Expertise erlangt.

6.9 Heterogenous Software Versions on KWB Computers

Unterschiedliche Softwareversionen (z.B. R) können dazu führen, dass Skripte auf verschiedenen Rechnern nicht das gleiche Verhalten zeigen.

Die IT-Abteilung ist in der Lage an bestimmte Nutzergruppen die gleiche Software (z.B. RStudio / R / Miktex) auszurollen. Dies sollte in Zukunft konsequent genutzt werden, indem auf alle Computer an denen potentiell programmiert wird zu dieser Nutzergruppe hinzugefügt werden und somit alle die gleichen Softwareversionen installiert haben.

6.10 R Package/Version Dependency of R Scripts

Lösungsvorschlag:

Es ist eine **Mindestdokumentation** der verwendeten R Version und sämtlicher R-Pakete (inklusive ihrer Versionen) zu fordern. Dazu kann in R die Funktion `sessionInfo()` genutzt werden. Die Ausgabe dieser Funktion kann entweder in eine Metadaten-Textdatei `session_info.txt` geschrieben werden oder im Falle der Erzeugung von R-Markdown-Dokumenten direkt am Anfang der Analyse im `R-Markdown` Dokument ausgegeben werden.

Das Schreiben der Metadaten-Datei `session_info.txt` sollte standardisert über eine Funktion in einem KWB R-Paket (z.B. `kwb.utils`) implementiert werden.

Direktausgabe in R Console / RMarkddown:

```
sessionInfo()
```

```
## R version 3.6.1 (2017-01-27)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.6 LTS
##
## Matrix products: default
## BLAS:   /home/travis/R-bin/lib/R/lib/libRblas.so
## LAPACK: /home/travis/R-bin/lib/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] kableExtra_1.1.0     tibble_2.1.3         knitcitations_1.0.10
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.2        highr_0.8         pillar_1.4.2
##  [4] compiler_3.6.1    plyr_1.8.4        tools_3.6.1
##  [7] zeallot_0.1.0     digest_0.6.21     viridisLite_0.3.0
## [10] lubridate_1.7.4   jsonlite_1.6      evaluate_0.14
## [13] pkgconfig_2.0.3   rlang_0.4.0       bibtex_0.4.2
## [16] rstudioapi_0.10   curl_4.1          yaml_2.2.0
## [19] xfun_0.9          RefManageR_1.2.12 httr_1.4.1
## [22] stringr_1.4.0     xml2_1.2.2        knitr_1.25
## [25] vctrs_0.2.0       hms_0.5.1         webshot_0.5.1
## [28] glue_1.3.1        R6_2.4.0          rmarkdown_1.15
## [31] bookdown_0.13     readr_1.3.1       magrittr_1.5
## [34] backports_1.1.4   scales_1.0.0      htmltools_0.3.6
## [37] rvest_0.3.4       colorspace_1.4-1  stringi_1.4.3
## [40] munsell_0.5.0     crayon_1.3.4
```

Schreiben in standardisierte Metadatei:

```
sink("session_info.txt")
sessionInfo()
sink()
```

Hierzu ist auch noch ein Tutorial zu erstellen!

Komplexere, technische Möglichkeiten zum Paketmanagement werden im gerade vom DFG gefördertem Projekt O2R in der Entwicklung befindlichen CRAN task view for computational environments and reproducibility genannt.

Mit dem R Paket packrat lässt sich das Paketmanagement ggf. verbessern.

Als Beispielanwendung dient die Datenanalyse zur Feinstaubbelastung mit Sensebox-Daten, die komplett reproduzierbar sind (mittels der Plattform mybinder) und ohne Installation von Interessierten im Webbrowser ausgeführt werden kann (siehe hier).

6.11 Complex R Script Dependencies

Dieses Problem tritt insbesondere auf, wenn mehrere verschiedene Nutzer gemeinsam mit den gleichen Skripten arbeiten (wie z.B. im abgeschlossenen Projekt OGRE).

Proposed solution:

Bewusstmachen der Skriptabhängigkeiten

Identifizieren von Optimierungspotentialen -> möglicherweise Elimination von Abhängigkeiten

Workflow dokumentieren und Tutorial, am besten als R-Markdown Dokument, erstellen. Das ist insbesondere wichtig, wenn Skripte häufig verwendet werden.

6.12 Heterogenous Coding Styles

Currently there is no established coding style at KWB for in case of programming e.g. R scripts

(R)Programmers at KWB will use the tidyverse coding style http:

//style.tidyverse.org as default. This will help increasing both, the readability and reusability of the developed (R-)scripts at KWB (currently: 1000 R scripts).

Recommended literature:

Clean code (Martin, 2009)

6.13 Collaborative Version Control

Multiple people developing code together.

1. At KWB we use Subversion (SVN) for version control (see: How to use subversion?)
2. Speak with each other (especially if you are working on code is likely to be changed be others)
3. Regularly perform updates/commits with the version control software SVN

In case the code base (e.g. R scripts) developed in 'project A' should be reused/adapted in a new 'project B' (e.g. FLUSSHYGIENE) it is not allowed to
change the code base in the original 'project A'. Instead the code needs to be copied in a folder for the new project by using Subversion's SVN copy command. In case this procedure is not followed established processes in the souce project will be overwritten.

6.14 Workflow Automation

Ressources:

- Reproducible Research Automation: This lesson shows how to use automation in R to improve the reproducibility of research by automating tasks.
- useR!2017 Video: Data Carpentry: Open and Reproducible Research (Tutorial)
  - Part1

  - Part2

- Use of an R package to facilitate reproducible research, e.g.:
  - rprrkg by rOpenSci
  - rrtools by Ben Marwick

Recommended literature:

- Kitzes et al. (2018)

6.15 Encoding

Umlaute und Sonderzeichen werden falsch angezeigt, wenn R-Skripte in unterschiedlichen Encodings abgespeichert und eingelesen werden.

Vorgabe einer Default Encoding Einstellung in RStudio (z.B. UTF-8)

Alternativ könnten auch alle Umlaute in Unicode dokumentiert werden (siehe folgendes Beispiel). Allerdings ist dies wohl nicht praktikabel, da die Lesbarkeit der Texte erschwert wird und es sollte daher von uns der erste Ansatz (Vorgabe von UTF-8 als Default Encoding) angestrebt werden.

A great blogpost for the topic (How do I write UTF-8 encoded content to a file?) is provided by Ushey (2018), including background information on how encoding in R works.

In order to increase the portability of R script he recommends the following:

> "Portable R scripts should use unicode code points, to avoid accidental mis-encoding of string literals."
>
> — (Ushey, 2018)

Encoding example in R:

```
### Richtiges Skript Encoding WICHTIG (richtiges Einlesen des Skripts nur wenn
### mit gleichem Encoding eingelesen  wie es auch abgespeichert wurde)
print("Ü")
```

```
## [1] "Ü"
```

```
### Skript Encoding EGAL (da Umlaut in Unicode codiert wurde)
print("\u00dc")
```

```
## [1] "Ü"
```

See also Hauke's article Understanding Encoding being part of the documentation of our R package kwb.fakin that is available on GitHub.

6.16 Collaborative Writing

Project proposal with many partners. How can be guaranteed that only the most recently document is used and how can multiple people work at the same time in that document in the "hot phase" of proposal writing?

Ulf made good experiences with Office 365 (provided by KWR at that time). Also writing of up to
10 persons in parallel were not a problem, because the currently processed paragraph automatically gets blocked for others.

However, in case a whole chapter needs to be processed by one person this is not performaed automatically. Thus it is recommended to delete the chapter for a short term from the syncronised document and replace it with a placeholder like 'Chapter currently in revision by XXXX. Will be put back in this document until YYYY-MM-DD'.

6.17 Exchanging Data with Colleagues/Project Partners

copied from: "10_old_german_chapters.Rmd", to be translated:

Es gibt internen Datenaustausch und Austausch mit Projektpartnern. Oft werden Dateien intern per E-Mail verschickt. Das sollten wir nicht tun, da wir unnötig Kopien von Dateien anlegen. Dateien sollten im besten Fall nur an einem Ort abgelegt sein. Dort sollten sie allerdings durch ein Backup-System gesichert sein. Natürlich muss der Zugriff auf diesen Ort gewährleistet sein. Einem Studenten mit eingeschränkten Rechten eine Datei per mail zuzuschicken, weil sie an dem Ort, an dem sie gespeichert ist, für ihn nicht zugänglich ist, ist meines Erachtens nicht die richtige Lösung. Es muss dann ein Ort geschaffen werden, an dem der gemeinsame Zugriff möglich ist. Laut Bodo ist der Ordner `Exchange` im Projektordner ein solcher Platz. Aus dem Grund der allgemeinen Zugänglichkeit wir er auch in manchen Projekten als allgemeiner Datenablageplatz "misbraucht". Es kann aber niemandem ein Vorwurf gemacht werden, da ja nirgendwo dokumentiert ist, was in diesem Ort abgelegt werden soll.

Das können wir nun ändern:

Laut Bodo sind Dateien im Exchange-Ordner nur kurzfristig abzulegen, um z.B. Studenten Zugriff darauf zu geben. Die Dateien sollen nach dem Gebrauch wieder aus dem Ordner gelöscht werden. Demnach sollte der Ordner im Normallfall leer sein. Dies ist in einigen Projekten nicht der Fall.

Wir brauchen eine einheitliche Definition der Bedeutungen von Ordnern.

This should be documented in the metadata.

Chapter 7

Glossary

7.1 Acronym

An acronym is a word or name formed as an abbreviation from the initial components of a phrase or a word. (Wikipedia)

7.2 Small research institute

Research institute with less than about 100 employees.

A detailed glossary covering the following topics is provided by Rokem and Chirigati (2018) is used throughout the glossary.

7.3 Reproducibility

> "… is a cornerstone of science. Definitions vary greatly across scientific disciplines, but the meaning that we find most prevalent is the 'calculation of quantitative scientific results by independent scientists using the original datasets and methods' (Stodden, Leisch, & Peng, 2014). The goals of reproducibility go beyond duplicating someone else's investigation: it also entails having reproducibility for yourself, defeating self-deception in scientific results (Ioannidis, 2005; Nuzzo, 2015), and extending another researcher's methods to build your own work. Reproducibility is a matter of degree, not of kind. We say that research is reproducible if reproducibility applies to the results to some extent. That is, some of the corresponding experiments and scientific methods are deemed to be reproducible.

— (Rokem and Chirigati, 2018)

For more details see: Rokem and Chirigati (2018)

7.4 Provenance

> "As the volume of digital data increases and the complexity of computational processes that manipulate these data grows, it is becoming increasingly important to manage their provenance. The Oxford English Dictionary defines provenance as the source or origin of an object; its history and pedigree; a record of the ultimate derivation and passage of an item through its various owners. Provenance helps determine the value, accuracy, and authorship of an object."
>
> — (Rokem and Chirigati, 2018)

For more details see: Rokem and Chirigati (2018)

7.5 Techniques

- Version control
- Literate Programming
- Data Publication
- Data cleaning/munging
- Software Testing
- Continuous Integration
- Workflow Management
- File Format Standards
- Licensing
- Virtualization and Environment Isolation

For details see: Rokem and Chirigati (2018)

7.6 Tools

- Programming Language and Related Tools
- Documentation Generators

- Version Control
- Data Munging and Analysis
- Data Visualization
- Software Testing and Continuous Integration
- Virtualization and Environment Isolation
- Data Sharing and Repositories
- Document Authoring
- File Formats

For details see: Rokem and Chirigati (2018)

Bibliography

(2019). Forschungslizenzen. http://forschungslizenzen.de/. Accessed: 2018-06-29.

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J., Starn, J., and Fienen, M. (2016a). Flopy: Python package for creating, running, and post-processing modflow-based models.

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J., and Fienen, M. N. (2016b). Scripting MODFLOW model development using python and FloPy. *Groundwater*, 54(5):733–739.

Boland, M. R., Karczewski, K. J., and Tatonetti, N. P. (2017). Ten simple rules to enable multi-site collaborations through data sharing. *PLOS Computational Biology*, 13(1):e1005278.

Bosman, J. and Kramer, B. (2016). Github and more: sharing data & code. https://101innovations.wordpress.com/2016/10/09/github-and-more-sharing-data-code/.

Bosman, J. and Kramer, B. (2018). Workflows. https://101innovations.wordpress.com/workflows.

Buettner, S., Hobohm, H.-C., and Mueller, L. (2011). *Handbuch Forschungsdatenmanagement*. Bock u. Herchen, Bad Honnef.

Fenner, M. and Haak, L. (2014). *Unique Identifiers for Researchers*, pages 293–296. Springer International Publishing, Cham.

Fenner, M., Scheliga, K., and Bartling, S. (2014). *Reference Management*, pages 125–137. Springer International Publishing, Cham.

Friesike, S. (2014). *Creative Commons Licences*, pages 287–288. Springer International Publishing, Cham.

Grolemund, G. and Wickham, H., editors (2017). *R for Data Science*. O'Reilly Media, Sebastopol, CA, 1 edition.

Hart, E. M., Barmby, P., LeBauer, D., Michonneau, F., Mount, S., Mulrooney, P., Poisot, T., Woo, K. H., Zimmerman, N. B., and Hollister, J. W. (2016). Ten simple rules for digital data storage. *PLOS Computational Biology*, 12(10):e1005097.

Kaden, B. and Kleineberg, M. (2018). Guidelines zur veröffentlichung dissertationsbezogener forschungsdaten.

Kitzes, J., Turek, D., and Deniz, F., editors (2018). *The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences*. University of California Press, Oakland, CA.

Kraus, F., Seis, W., Remy, C., Rustler, M., Jubany i Guell, I., Espi, J. J., and Clarens, F. (2016). Deliverable d3.2: Show case of the environmental benefits and risk assessment of reuse schemes. Report, public report in the DEMOWARE project (www.demoware.eu), KWB.

Martin, R. C. (2009). *Clean Code*. Pearson Education, Inc. ISBN-13 978-0-13-235088-4978-1498716963.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rokem, A. and Chirigati, F. (2018). *Glossary*. University of California Press, Oakland, CA.

Rustler, M. (2016a). kwb.hantush (v.0.2.1). Tutorial: http://kwb-r.github.io/kwb.hantush/.

Rustler, M. (2016b). kwb.qmra (v.0.1.1).

Rustler, M. (2016c). Quantitative microbiological risk assessment for different wastewater reuse options in Old Ford (v.1.0). For performing the quantitative microbial risk assessment the R package kwb.qmra, v.0.1.1 (https://doi.org/10.5281/zenodo.154111) is used.

Rustler, M. (2018). aquanes.report (v.0.5.0).

Sonnenberg, H. (2018). kwb.logger (v 0.2.0).

Sonnenberg, H. and Rustler, M. (2016). kwb.wtaq (v.0.2.1). Tutorial website: http://kwb-r.github.io/kwb.wtaq.

Stodden, V. (2014). *Intellectual Property and Computational Science*, pages 225–235. Springer International Publishing, Cham.

Taschuk, M. and Wilson, G. (2017). Ten simple rules for making research software more robust. *PLOS Computational Biology*, 13(4):e1005412.

Ushey, K. (2018). String encoding and r. https://kevinushey.github.io/blog/2018/02/21/string-encoding-and-r/.

Wickham, H., editor (2015a). *Advanced R*. The R Series. Chapman and Hall/CRC, Boca Raton, FL, 1 edition.

Wickham, H., editor (2015b). *R Packages: Organize, Test, Document, and Share Your Code*. O'Reilly Media, Sebastopol, CA, 1 edition.

Wickham, H., editor (2018). *Advanced R*. 2 edition.

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., and Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6):e1005510.