Technical Solutions for Reproducible Research

Alexander König
Eurac Research, Italy /
CLARIN ERIC, the Netherlands
alex@clarin.eu

Egon W. Stemle
Eurac Research, Italy
Egon.Stemle@eurac.edu

André Moreira
CLARIN ERIC, the Netherlands
andre@clarin.eu

Willem Elbers
CLARIN ERIC, the Netherlands
willem@clarin.eu

Abstract

In recent years, the reproducibility of scientific research has increasingly come into focus, both by external stakeholders (e.g. funders) and by the research communities themselves. Corpus linguistics, with its methods for creating, processing and analysing corpora, is an integral part of many other disciplines that work with language data and therefore plays a special role. Moreover, language corpora are often living objects that are regularly improved and revised. At the same time, tools for the automatic processing of human language are also being developed further, which can lead to different results with the same processing steps and the same data. This article argues that modern software technologies, such as version control and containerisation, can mitigate the following problems: Software packaging, installation and execution and, equally important, the tracking of corpus modifications throughout its life-cycle. All in all, this leads to transparency of changes to raw data and software tools and thereby enhanced reproducibility.

1 Introduction

While reproducibility has always been one of the main pillars of scientific research, within the last ten years, this has come even more into focus for the social sciences and humanities. Prominent cases of scientific fraud, for example, the case of Diederik Stapel in the Netherlands (Levelt et al., 2012), have brought problems about the reproducibility of scientific research into focus. In this article, we discuss possible techniques to handle this problem by using standard tools from the realm of software development. We propose to use versioning software to ensure the persistence of data (see section 2) and containerisation to ensure the same for natural language processing (NLP) tool-chains (see section 3), both concepts are illustrated with case studies (see section 2.2 and section 3.2) and finally, we highlight some challenges we encountered along the way (see section 4).

2 Ensuring persistence of data

2.1 Introduction

After the initial data collection, a corpus usually keeps evolving while initial analyses are already being carried out. It is also likely that while working on the corpus and analysing the data, errors in the transcriptions or annotations are discovered, which need to be corrected. And with a rich annotation scheme that is constantly being re-evaluated and refined this is usually all the more true. Likewise, there are usually changes to the tools and the processing pipelines used for the processing of the corpus and involved in its analysis. More generally, the different research strategies of the method-oriented computational linguistics, and the hermeneutic tradition of the social sciences and humanities (SSH) may contribute to systemic iterations of the tools and their processing pipelines, and the corpus data and its assessment strategies (Kuhn, 2019).

While these kind of changes are unproblematic as long as the corpus is still in its ”building phase”, as soon as the first analyses have been made public, any change to the data will endanger the possibility of

reproducing these analyses. Therefore, the researchers have to preserve a version of the corpus exactly as it was when a specific analysis was made, and a snapshot of all tools that were used to process the corpus and were involved in its analyses must be available in order to ensure *reproducibility*.

Following Cohen et al. (2018), we assume that such availability of tools and data will help to achieve two of the three dimensions of *reproducibility* they mention. They introduce 'Three Dimensions of Reproducibility in Natural Language Processing': 1. 'Reproducibility of a conclusion', 2. 'Reproducibility of a finding', and 3. 'Reproducibility of a value'. By *conclusion* they mean "a broad induction that is made based on the results of the reported research, by *finding* they mean "a relationship between the values for some reported figure of merit with respect to two or more dependent variables", and by *value* they mean "a number, whether measured (e.g. a count of false positives) or calculated (e.g. a standard deviation)". Broadly speaking, availability of the original data gives access to the core values (numbers) of corpora, and access to the state of tools enables the processing and comparing of numbers to discover the same relations between the numbers. The more abstract conclusions are not affected.

Using version control systems (VCSs) to provide "a lightweight yet robust framework that is ideal for managing the full suite of research outputs such as datasets, statistical code, figures, lab notes, and manuscripts" has already been suggested by Ram (2013) for bio medicine; Stodden et al. (2018) (first printed edition from 2014) have a very comprehensive overview from computational science; Nüst et al. (2018) give an overview of reproducibility in the field of geographic information science (GIS) and are careful to include information on their paper's reproducibility, for which they publish data, code, and a description of the runtime environment[1]; and Brinckman et al. (2019) introduce the Whole Tale project that connects "computational, data-intensive research efforts with the larger research process – transforming the knowledge discovery and dissemination process into one where data products are united with research articles to create *living publications* or tales" with application in material science, astronomy, and archaeology. Within the SSHs, for example, the Center for Reflected Text Analytics (CRETA)[2] "focuses on the development of technical tools and a general workflow methodology for text analysis within Digital Humanities. Of particular importance is the transparency of tools and traceability of results, such that they can be employed in a critically-reflected way." For NLP, projects like LaMachine[3] bundle numerous open-source NLP tools, programming libraries, web-services, and web-applications in a single Virtual Research Environment with the possibility to install explicitly defined versions for all software to support scientific reproducibility. In the following, we will make our proposal to further stimulate discussion in the community – inspired by the CLARIN infrastructure, but keeping in mind a general validity beyond it.

If the corpus in question is a text corpus (probably being stored in some kind of XML format like e.g. TEI[4]), an obvious solution to these problems is to use existing VCSs like subversion[5] or git[6] to keep track of all changes within the corpus.

This is also possible for corpora that are not mainly text-based, for example, multimodal corpora. First, they often have a text-component in their annotations (which could have been done, for example in ELAN[7] or EXMARaLDa[8], both of which store their data in XML format) and it is usually this part that will change the most within the lifetime of such a corpus, while the primary audio/video data often remains untouched. Secondly, although the problem of storing large files in versioning tools is grave, there will likely be solutions in the foreseeable future. Using such an existing versioning software solution, all changes throughout the life cycle of a corpus can be tracked, and through the use of code hosting platforms like Github[9] or GitLab[10] all changes can be made visually appealing to the research community as a

[1] https://doi.org/10.5281/zenodo.1227260
[2] https://www.creta.uni-stuttgart.de/en/
[3] https://proycon.github.io/LaMachine/
[4] https://tei-c.org/
[5] https://subversion.apache.org/
[6] https://git-scm.com/
[7] https://tla.mpi.nl/tools/tla-tools/elan/
[8] https://exmaralda.org/en/
[9] https://github.com/
[10] https://www.gitlab.com

whole.

Having the corpus available on an online repository platform, while having the advantage of being very transparent about all changes made to the data, might not be the ideal way of providing the data to other researchers. Therefore traditional data repositories like CLARIN Centres, META-SHARE[11], and zenodo[12] will still play a role in making the data available to the users and especially in providing findability (through participation in search interfaces like the VLO[13] or the OLAC catalogue[14]) and issuing persistent identifiers to specific versions.

2.2 Case Study: The DiDi Corpus

The Institute for Applied Linguistics (IAL) at Eurac Research is currently investigating how it can move towards such a setup for more reproducibility in research as outlined in the previous section. One of the first corpora that was transformed into such a strictly versioned environment is the DiDi corpus (Frey et al., 2016). The corpus is available under an academic non-commercial (ACA-NC) license from an on-premise GitLab installation[15]. The whole corpus data is divided into multiple parts for the different file formats in which the corpus is available and is accompanied by extensive documentation. The different versions of the corpus are realised as tags in GitLab, which are references to the full state of the data repository at different times during its development. Additionally, these tagged versions are also uploaded into the Eurac Research CLARIN Centre (ERCC), the CLARIN DSpace repository hosted by the IAL, so they can be easily downloaded by less tech-savvy users[16]. Another advantage is, of course, that this integration of the data into a CLARIN Centre will make the metadata available to various search engines (e.g. the VLO or the OLAC search) and it can therefore be discovered more easily. All the data for a tagged version is available both at the ERCC and on GitLab with each of these hosting platforms cross-referencing the other. At both places, all versions are accompanied by a changelog that explains the changes between versions. On GitLab, the interested user can also make use of the integrated version diff to get more fine-grained information on the changes between versions.

However, in the attempt to implement the paradigm for explicit versioning of corpora, the DiDi corpus has a feature that requires special attention: the corpus contains personal information for which the corpus creators have asked the users for their consent to share the data, and this consent was explicitly requested for re-use in academic contexts.

More generally, linguistic corpora often consist of personal data produced by individuals where both privacy and IPR concerns need to be considered. And if not all of the data can be made publicly available, there has to be additional access protection both on the side of the DSpace repository and on the side of GitLab. While it is easy to have some data require a login with an academic account (for example, by using the CLARIN federated login[17]) in DSpace, the GitLab repository should ideally not be made completely password protected, but have at least an openly available landing page that describes the corpus. At the IAL this has been implemented for the DiDi corpus using git submodules where the main repository with the documentation and the overview of the various data formats is publicly accessible and the actual data is in sub repositories that require a login[18]. Still, all license information and documentation is available *without* login (see Figure 1). It is likely that more complex access scenarios will prove even more difficult to map to a code hosting platform.

[11] http://www.meta-share.org
[12] https://zenodo.org
[13] https://vlo.clarin.eu/
[14] http://search.language-archives.org
[15] https://gitlab.inf.unibz.it/commul/didi
[16] https://hdl.handle.net/20.500.12124/7
[17] https://www.clarin.eu/content/federated-identity
[18] https://gitlab.inf.unibz.it/commul/didi/data-bundle

```
data-bundle/
├── CHANGELOG.html
├── CHANGELOG.md
├── data-annis/    @ecfce535
│   (git submodule with restricted access)
├── data-didijson/ @e9077a17
│   (git submodule with restricted access)
├── data-didixml/  @f7eb581d
│   (git submodule with restricted access)
├── data-docs/     @47299aef
│   ├── CHANGELOG.md
│   ├── DiDi_annotation_layers_DE.pdf
│   ├── DiDi_annotation_layers_EN.pdf
│   ├── DiDi_anonymisation_DE.pdf
│   ├── DiDi_anonymisation_EN.pdf
│   ├── DiDi_cmc_annotations_DE.pdf
│   ├── DiDi_cmc_annotations_EN.pdf
│   ├── DiDi_metadata_DE.pdf
│   ├── DiDi_metadata_EN.pdf
│   ├── LICENSE
│   └── README.md
├── EULA-CLARIN-ACA-BY-NC-NORED.md
├── EULA-CLARIN-ACA-BY-NC-NORED.pdf
├── EULA-CLARIN-ACA-BY-NC-NORED.txt
├── LICENSE
├── Makefile
├── README_gen.sh
├── README.html
└── README.md
```

Figure 1: Directory structure of the DiDi repository on GitLab

3 Methods and tools and their impact on reproducibility

3.1 Introduction

In linguistic research – especially in the sub-fields of corpus linguistics and natural language processing – data is often processed with the use of quite intricate software tool-chains. Ranging from more simple tasks like tokenisation or lemmatisation to more complicated ones like fine-grained syntactic parsing. The unification of all the necessary tools for the automatic processing of human language into an integrated processing framework is more the exception than the norm. This inevitably leads to a wide variety of individual solutions each with their own installation procedures, development life cycles with maintenance and update schedules, etc. (Wieling et al., 2018). Furthermore, linguistic models that are often at the heart of such tools are also subject to change, and this change need not necessarily be synchronised with the tool itself, spanning an even wider range of possible combinations (see e.g. (Nothman et al., 2018)).

This short overview already shows the difficulty for other researchers to exactly recreate a certain tool-chain to verify research results. It can only be ensured if the original researchers document their setup carefully, noting down exactly which version of a certain tool was used and how exactly the various tools were combined. An additional problem is that some software manufacturers do not make older versions of their products easily available, so even if the version is known it is not certain that it can be obtained when necessary. For this reason it has been discussed whether scientific software should be archived in research repositories alongside the data, but so far, while some CLARIN repositories do also host linguistic tools, little progress has been made in this regard.

The recent trend in software deployment and administration towards containerisation of services seems

to us to be a promising solution to the aforementioned problems regarding the reproducibility of data processing in linguistic research.

Containerisation means that programs are not installed on a real computer or in a full-blown virtualised environment like a virtual machine, but instead in a very reduced environment that leaves out everything that is not vital for the program in question to work. The idea is to minimise both the amount of memory and processor time needed for such a containerised service and especially the possibility for unwanted side effects. With Docker[19] this way of packaging programs and services has been widely adopted within the last years and the additional possibility to orchestrate the deployment of such minimal containers using a platform like Kubernetes[20] makes using existing containers "off the shelf" quite easy, especially because a lot of the big infrastructure providers (e.g. Google[21], Microsoft[22] or Amazon[23]) offer ways to deploy containers on their infrastructure for a moderate price and there seems to be a trend to make this kind of deployment as easy as possible[24].

As a researcher, building the tool-chain for a new project can be done directly in Docker. There are already a variety of places where the resulting docker images (from which various container instances can be created) can be stored for re-use by others. For example, GitLab offers such a Docker image registry. GitLab is also a place where the data can be stored (see section 2), and both the data and the tool-chain used could thus be stored in one place, making it much easier for researchers planning to recreate an experiment to get both in exactly the same versions that had been used originally. The wide availability of container hosting (see above) also means that it will be quite easy to simply take such a container with the whole tool-chain setup and use it to verify the results or look for something else in another set of data while ensuring that the same methodology is used as in the original research.

3.2 Case Study: Gitlab and Docker - A framework for reproducible research

CLARIN ERIC has been working on a technical solution to manage their infrastructure based on docker images and a set of scripts to build, test and run these images (Elbers et al., 2019). It turns out this approach is also quite useful to tackle a number of the issues with respect to reproducibility of research. The core concepts of the approach are (1) manage the container image definition (Dockerfile when using docker) in a git repository, (2) use tags and a continuous integration (CI) solution to build and publish the resulting container image, preferably into a publicly accessible container registry. These initial requirements have since been extended, to accommodate the reproducibility of research, with (3) a clear container image template as shown in listing 1, (4) a clearly specified input format which should be verified before running the analysis and (5) a predefined command and set of arguments to run the image as a container and collect its output as shown in listing 2. Altogether this ensures a workflow where the same set of tools can be run, in an environment that is guaranteed to be stable, on different inputs (as long as the input specification is followed) to produce comparable results. This has been documented on GitLab [25].

```
1  #Base this image on Alpine Linux to ensure a small footprint to
2  #start with.
3  FROM alpine:3.11
4
5  #Install or compile any required packages and/or scripts here
6
7  #Define input directory, this will be populated with data via a host
8  #mount (see listing 2 Docker run command)
9  VOLUME "/input"
10 #Define output directories, this is where the /run.sh should produce
11 #all results and this data is exported to the host system via a host
```

[19] https://www.docker.com/
[20] https://kubernetes.io/
[21] https://cloud.google.com/kubernetes-engine/
[22] https://azure.microsoft.com/en-us/services/kubernetes-service/
[23] https://aws.amazon.com/containers/
[24] https://cloud.google.com/blog/products/serverless/introducing-cloud-run-button-click-to-deploy-your-git-repos-to-google-cloud
[25] https://gitlab.com/CLARIN-ERIC/reprolang

```
#mount (see listing 2 Docker run command)
VOLUME "/output/datasets"
VOLUME "/output/tables_and_plots"

#Add and set an executable entrypoint script. This script _must_ be
#implemented by the researcher, can call other scripts and is
#responsible for:
#1. Ensure input data is available in /input
#2. (Optional) Verify input
#3. Process all input data so it is ready for analysis (e.g. extract
#   tarball)
#4. Run analysis on the input data
#5. Produce final result in /output/datasets and
#   /output/tables_and_plots
ADD run.sh /run.sh
RUN chmod u+x /run.sh
ENTRYPOINT /run.sh
```

Listing 1: Dockerfile template example

```
docker run \
    -ti --rm --name=${PROJECT_NAME} \
    -v ${PWD}/input:/input \
    -v ${PWD}/output/datasets:/output/datasets \
    -v ${PWD}/output/tables_and_plots:/output/tables_and_plots \
    <image name>:<image tag>

#Explanation:
#Line 1 instructs docker to start a container
#Line 2 sets an interactive TTY (-ti), will remove the container
#       when done (--rm) and give the container a name (--name)
#Line 3 configures what directory on the host to mount into /input
#       inside the container.
#Line 4 and 5 configure what directories on the host to mount into
#       /output/datasets and /output/tables_and_plots
#Line 6 specifies which image and which tag of that image to start
```

Listing 2: Docker run command

Based on the experience in using this approach to run the CLARIN ERIC production environment for more than two years, the following best practices are advised: (1) always use version pinning for any packages installed into the container image. This will ensure the image build will break if there is an issue with any of the packages, providing a clear signal something will be different in the environment compared to earlier runs. The integration with a CI pipeline, and specifically the GitLab CI pipeline as described in (Elbers et al., 2019), ensures that an image is created as soon as a new version (git tag) is released. This image provides the exact, tagged and verifiable (commit hash), run time environment to repeat the experiment as long as the image is available, even if the image build is failing because of package version, or other, issues. In addition to pinning versions in the container image, including scripts and dependencies at run time (via mounts) should also be avoided since it directly violates the requirement to embed all tools and packages into the container image.

Some other more generic best practices with respect to container images: try to reduce the resulting image size, e.g. when using Docker base your image on the Alpine Linux [26] image (alpine:3.11 [27] equals approximately 5.59MB) rather than the Ubuntu [28] image (ubuntu:18.04 [29] equals approximately 64.2MB), try to reduce the number of layers by grouping related commands, e.g. using one `RUN` statement and chain the commands using the `&&` operator.

Using the framework as described so far also requires the researchers to adopt a new workflow to produce results. Building this container image is not a step one has to perform after the fact, just before

[26] https://alpinelinux.org/
[27] https://hub.docker.com/_/alpine/
[28] https://ubuntu.com/
[29] https://hub.docker.com/_/ubuntu/

publishing the results. Rather this framework and the introduced best practices should be included into the research life cycle from the beginning and all results published should be obtained by running a properly tagged and published container image, run exactly according to the input specification. This guarantees that as long as the container image is available from the public container image registry and the dataset is available, the research can be reproduced. As long as the container image is available, the whole tool chain can also be used against different inputs without any additional effort. If, for whatever reason, the container image is not available anymore, the Dockerfile can be used to rebuild the container image. If any of the required dependencies are no longer available, the Dockerfile will have documented all required dependencies with explicit versions, commands and command line arguments. We advocate to take over all data management best practices that have been developed in the area of long term preservation for research data for container images as well.

4 Challenges and Pitfalls

Some of the possible problems that can be encountered were already addressed within the case studies (see section 2.2 and section 3.2), but we want to highlight them here again.

On the data side the most important problem that we encountered is that linguistic data is, by its very nature, personal data, which means that there will always be privacy concerns when sharing language corpora with the wider research community. Protecting data from some people while giving access to others is relatively easy on the side of a research data repository - after all that is one of their key functions - but it proves to be a bit more difficult on the side of a code hosting repository, where there normally is no need for fine-grained access permissions. The use of git submodules as described in section 2.2 seems to be an efficient solution to this problem, but it remains to be seen how well this works for more complicated authorisation scenarios.

There is also, as always when using external services for sensitive data, the consideration whether one should store data with a commercial provider, especially one based in another country and jurisdiction. Apart from privacy concerns, there is also the possibility that a commercial provider suddenly goes out of business or decides to move away from this type of service. One way to avoid these problems is the GitLab "community edition"[30] that is avaible as open source and can be installed on local infrastructure, meaning that the researcher/the institute will be able to keep full control of the data and container hosting.

On the technological side, there are also a few problems that need to be addressed. For example, `git` has just arrived at its third iteration of dealing with large files: First, `git annex`, then `git lfs`, and now `git` partial `clone`[31]. This means that the technological barrier to deal with large files in one of the more common VSCs has – at best – just been made a tad smaller, but whether this latest iteration will finally solve the problem remains to be seen. Of course, there exist more mature solutions to this problem, like Data Science Version Control (DVC)[32] that extends `git`'s functionality with explicit support for large files, but this has the disadvantage of introducing another git-like-tool into the mix. Albeit `git`'s popularity, if even "more experienced Git users requested that someone else perform an operation" because they were scared (Church et al., 2014), it is certainly no exaggeration to say that one of the very important reasons for `git`'s success is "[i]n a word, GitHub" [33] or, more generally, the online platforms. Overall, the online repository hosting platforms make many tasks easier while dealing with the intricacies of `git`, and for any additional tool, especially one that bears similarities with `git`, there is a need for such aid.

Another possible pitfall lies in the use of Dockerfiles to create a persistent setup of a tool-chain. Unfortunately, when writing a Dockerfile there is currently no enforcement of explicit versioning of the docker software used with the Dockerfile. This means that two containers built using the same Dockerfile at two different points in time can contain two slightly different versions of the software which may result in different behaviour. The software specifications within the Dockerfiles rely on versioned resources from

[30] `https://about.gitlab.com/install/ce-or-ee/`

[31] `https://about.gitlab.com/blog/2020/03/13/partial-clone-for-massive-repositories/`

[32] `https://dvc.org/`

[33] `http://blogs.nature.com/naturejobs/2018/06/11/git-the-reproducibility-tool-scientists-love-to-hate/`

the software builder. However, these are not always available. This is even true for the base images (e.g. Ubuntu) hosted at Docker Hub. They are constantly updated with the latest patches which might result in different behaviour. Accessing older versions of e.g. the Ubuntu image on Docker Hub is very difficult to impossible. A similar problem exists for the container image registry, where there is not necessarily an explicit connection between the image and the Dockerfile (version) that has been used to create it. All images exist more or less on their own and there is no mechanism with which one can mark an image as a successor or predecessor of another image.

The framework for reproducible research discussed in section 3.2 provides a solution to this challenge of making sure tool versions do not differ, as long as proper data management is applied to the container images. While the container image is accessible, guarantees on stability of the environment can be made and there should be no differences in versions of software. Only if the container image is lost or no longer accessible, these guarantees can no longer be made.

The biggest challenge to us, however, seems to be to change the workflow of researchers to incorporate this way of doing research into their projects from the beginning. Only if both data and processing software are properly managed from the very start of a new project, can it be assured that the research that is conducted within the project will be reproducible by other researchers later on. Especially the need for very explicit templates and usage guidelines as described in section 3.2 needs to be well-thought-out and then followed precisely.

5 Conclusion and Outlook

Reproducibility of corpus-linguistic research is a central problem within the linguistic community (Wieling et al., 2018) which is currently not well addressed in a large number of projects. In this paper, we have presented a promising approach to tackle this problem using existing tools from the software development world. We have showed some case studies where this approach has been implemented, but already encountered a number of potential problems of which we highlighted some. Nevertheless, this way of ensuring that results in corpus-based linguistic research can easily be reproduced by fellow researchers seems like an idea that is worth pursuing in the future.

CLARIN ERIC is already working on establishing best practices and a framework that can be used to improve the reproducibility of research in the way described in this paper. At the time of writing, a pilot test of this framework is ongoing in the scope of the Language Resources and Evaluation Conference (LREC) 2020[34], where it provides the technical basis for the REPROLANG 2020[35] shared task. Thereafter, CLARIN ERIC will publish a follow up article providing a thorough analysis of the pilot results and suggesting further improvements based on this concrete experience. The proposed framework and best practices are all based on open source technologies and public service offerings for publicly available data. Proper exit strategies can be put in place to deal with worst case scenarios where some of these services might disappear or become unusable for whatever reason. We are confident that these exit strategies can be implemented for all services used in the described approach.

References

Adam Brinckman, Kyle Chard, Niall Gaffney, Mihael Hategan, Matthew B. Jones, Kacper Kowalik, Sivakumar Kulasekaran, Bertram Ludäscher, Bryce D. Mecum, Jarek Nabrzyski, Victoria Stodden, Ian J. Taylor, Matthew J. Turk, and Kandace Turner. 2019. Computing environments for reproducibility: Capturing the "Whole Tale". *Future Generation Computer Systems*, 94:854–867, May.

Luke Church, Emma Söderberg, and Elayabharath Elango. 2014. A case of computational thinking: The subtle effect of hidden dependencies on the user experience of version control. In Benedict du Boulay and Judith Good, editors, *Proceedings of the 25th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2014, Brighton, UK, June 25-27, 2014*, page 16. Psychology of Programming Interest Group.

K. Bretonnel Cohen, Jingbo Xia, Pierre Zweigenbaum, Tiffany J. Callahan, Orin Hargraves, Foster Goss, Nancy Ide, Aurélie Névéol, Cyril Grouin, and Lawrence E. Hunter. 2018. Three Dimensions of Reproducibility in

[34] https://lrec2020.lrec-conf.org/

[35] https://lrec2020.lrec-conf.org/en/reprolang2020/

Natural Language Processing. *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, pages 156–165, May.

Willem Elbers, Egon W. Stemle, André Moreira, Alexander König, Luca Cattani, and Martin Palma. 2019. The clarin eric deployment infrastructure and its applicability to reproducible research.

Jennifer-Carmen Frey, Aivars Glaznieks, and Egon W. Stemle. 2016. The DiDi Corpus of South Tyrolean CMC Data: A multilingual corpus of Facebook texts. In Pierpaolo Basile, Anna Corazza, Franco Cutugno, Simonetta Montemagni, Malvina Nissim, Viviana Patti, Giovanni Semeraro, and Rachele Sprugnoli, editors, *Proceedings of Third Italian Conference on Computational Linguistics (CLiC-it 2016) & Fifth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2016)*, Napoli, Italy, December.

Jonas Kuhn. 2019. Computational text analysis within the Humanities: How to combine working practices from the contributing fields? *Language Resources and Evaluation*, 53(4):565–602, December.

Willem JM Levelt, PJD Drenth, and E Noort. 2012. Flawed science: The fraudulent research practices of social psychologist diederik stapel.

Joel Nothman, Hanmin Qin, and Roman Yurchak. 2018. Stop Word Lists in Free Open-source Software Packages. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 7–12, Melbourne, AU, July. Association for Computational Linguistics.

Daniel Nüst, Carlos Granell, Barbara Hofer, Markus Konkol, Frank O. Ostermann, Rusne Sileryte, and Valentina Cerutti. 2018. Reproducible research and GIScience: an evaluation using AGILE conference papers. *PeerJ*, 6:e5072, July.

Karthik Ram. 2013. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, 8(1):7, February.

Victoria Stodden, Friedrich Leisch, and Roger D. Peng, editors. 2018. *Implementing Reproducible Research*. Chapman and Hall/CRC, 1 edition, December.

Martijn Wieling, Josine Rawee, and Gertjan van Noord. 2018. Reproducibility in Computational Linguistics: Are We Willing to Share? *Computational Linguistics*, 44(4):641–649, December.