D7.3

On Versioning Living and Programmable Corpora

(Executable) Report and Prototypes for Reproducible Research

Authors of the Report: Ingo Börner, Peer Trilcke

Concept & Development of the DraCor Prototype: Frank Fischer, Carsten Milling – Ingo Börner, Mathias Göbel, Mark Schwindt, Daniil Skorinkin, Henny Sluyter-Gäthje, Peer Trilcke

Date: February 27, 2024

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004984

Project Acronym: CLS INFRA
Project Full Title: Computational Literary Studies Infrastructure
Grant Agreement No.: 101004984

Deliverable/Document Information

Deliverable No.: D7.3
Deliverable Title: On Versioning Living and Programmable Corpora
Authors: Ingo Börner, Peer Trilcke
Review: Henny Sluyter-Gäthje, Maciej Eder, Daniil Skorinkin, Frank Fischer
Dissemination Level: PUBLIC

Document History

Version/Date	Changes/Approval	Author/Approved by
2024-02-22	Version for Review	Ingo Börner, Peer Trilcke
2024-02-27	Version for Submission	Ingo Börner, Peer Trilcke

Index

List of Figures

List of Tables

0. About this Deliverable

This report is accompanied by a web-based version that includes executable (and therefore fully reproducible) versions of the analyses in chapter 3. This executable report can be accessed at: https://github.com/dh-network/clsinfra-d73

In addition to this report (Deliverable D7.3) includes a technical prototype for Programmable Corpora: the Drama Corpora Platform "DraCor", which has been under construction since 2018 and was most recently further developed within CLS INFRA.
The main access points to the different components of the DraCor system are:

- Code and Data on GitHub: https://github.com/dracor-org
- DraCor Front-end: https://dracor.org/
- DraCor API: https://dracor.org/doc/api

Numerous scholars participate in the development of DraCor and in the curation of the corpora (the latter is not part of this Deliverable). For an overview, see https://dracor.org/doc/credits.
For the DraCor platform as a whole, these persons are responsible:

- Editor-in-chief: Frank Fischer (Freie Universität Berlin)
- Co-editors: Peer Trilcke (University of Potsdam), Julia Jennifer Beine (Ruhr University Bochum), Daniil Skorinkin (University of Potsdam)
- Technical lead: Carsten Milling (University of Potsdam)
- Technical co-leads: Ingo Börner (University of Potsdam), Mathias Göbel (University of Göttingen)
- Tools: Henny Sluyter-Gäthje (University of Potsdam)
- Art director: Mark Schwindt (Freie Universität Berlin)

1. Publishable Summary

Digital corpora, which are proving more and more to be the most important epistemic objects of Computational Literary Studies (CLS), are by no means always static objects. On the contrary, it is becoming increasingly clear that the digitisation of our cultural heritage needs to be understood as an ongoing process, which also implies that a number of the epistemic objects of CLS must be conceptualized as genuinely dynamic. We address this specific quality of some epistemic objects of the CLS by speaking of "living corpora". Where corpora — as the *data* of CLS — are also conceptually combined with *code* (e.g. in the form of an API) to form more complex research artifacts, we speak of "programmable corpora", as described in detail in CLS INFRA Deliverable D7.1 "On Programmable Corpora".

However, both living and programmable corpora usually face a considerable problem when discussed with regard to the reproducibility of research. This report considers possible solutions for the stabilization of living and programmable corpora and thus shows ways of making them available for reproducing research in a sustainable and long-term manner.

By recommending Git commits as a way for versioning living corpora, we rely on a well-established and proven tool for distributed version control, which, as we show using a concrete example, can also be used for living corpora. This also offers the possibility of retrieving additional (technical and performative) metadata about corpora.

For the more complex programmable corpora, on the other hand, we recommend the containerization of the entire research infrastructure.

In a broader sense, this report is also an exploration of the traces left by a living corpus in the technical space of a Git-based version control system. The traces are recovered using a method that we call “algorithmic corpus archaeology” – a method which we recommend to all those who embark on the epistemological adventure of working with living and programmable corpora.

2. Introduction: The Problem of Reproducibility in Dynamic Digital Ecosystems

angels sing, and a light
suddenly fills the room.

Linus Torvalds, "git/README", April 8, 2005

commits shape history

Git Guides, "Git commit", October 29, 2021

Reproducibility has never been a central methodological problem for traditional literary studies. Certainly, an interpretation of a poem (for example) should be comprehensible, its arguments plausible, its evidence empirical. But hardly anyone demands that such an interpretation should be "reproducible", let alone by a researcher other than the one who originally provided the interpretation.

Actually, the claim for reproducibility only enters the field of literary studies when empirical methods are adopted, for example from sociology or psychology. Thus, with the rise of Computational Literary Studies (CLS), which are also committed to an empirical methodology, a new, quite wide field of research has recently opened up in which literary studies are confronted with the problem of reproducibility. Yet "repetitive research" (to use a broader term, following Schöch 2023) can take very different forms: One might think of the *re-implementation* of methods and scripts in new research projects; of the *re-analysis* of data sets with optimized tools; or of the exact *re-production* of analyses in the course of scientific quality assurance, for example in peer review. In these and many other respects, Computational Literary Studies (but also Computational Humanities and Digital Humanities in general) are facing the demand for reproducibility.

However, according to critical voices, research in the humanities has not adequately met this demand. Alluding to the so-called "replication crisis" (Open Science Collaboration 2015) in some empirical sciences (particularly in psychology and medicine), James O'Sullivan, for example, stated in 2019 that "the humanities have a 'reproducibility' problem" (O'Sullivan 2019). In her widely discussed critique of CLS, Nan Z Da pointed out that in several cases it was not possible to reproduce the results of research in this field (Da 2019). And in a paper as relevant as it is comprehensive, Christof Schöch recently concluded that when it comes to "reproducibility" there are "serious and relevant challenges for the field of CLS", "starting with issues of access to data and code, but also concerning questions of lacking reporting standards, limited scholarly recognition, and missing community commitment and capacity that would all be needed to foster a culture of [repetitive research] in CLS and beyond" (Schöch 2023: 379).

This report particularly addresses one aspect of the far-reaching disciplinary reproducibility challenge, namely the *stabilization of living (and programmable) corpora*. Before we address this aspect, we need to briefly explain what we mean when we use the term “living corpora” (in reference to terms such as “living document”, see Shanahan 2015): Central to any form of reproductive research is the object to be researched, let's call it the epistemic object. In CLS, this epistemic object is regularly no longer just an individual text or a small group of individual texts, but a “corpus” and thus an entity that — “across many research domains in the humanities and social sciences” — “has emerged as a major genre of cultural and scientific knowledge” (Gavin 2023: 4).[1] Now, there is a large number of corpora that can be fully digitized with manageable resources, for example authors' corpora (such as all of William Shakespeare's comedies or all of Henrik Ibsen's plays). On the other hand, there is also a large number of epistemic objects, i.e. CLS corpora, which cannot be made digitally available so easily. In many cases, we don't even know exactly which texts would have to be included in such corpora. Not to mention that some texts aren’t available in digital form. In all these cases, we must assume that the epistemic object of CLS is currently (and presumably for a long time to come) *in the making* – in the process of becoming, of growing and thus, in a certain sense, “living”. Therefore, speaking of "living corpora" emphasizes that the digitization of our cultural and literary heritage is not so much a state that is or could be achieved, but rather a process, a (permanent) mode of transformation that we have entered. One of the consequences is that these epistemic objects of CLS must be conceptualized as dynamic.[2]

We address this dynamic nature of the literary data in this report. What we have called the problem of the "stabilization of living corpora" can be understood — as the title of our report suggests — as a versioning task: If there is a comprehensive, transparent and fully addressable versioning mechanism for our dynamic epistemic object, then stabilization can be achieved by pointing to a particular version. Furthermore, if corpora are coupled with lightweight research infrastructures in the form of research-driven APIs, as in the case of "Programmable Corpora" (a concept introduced in Fischer et al. 2019; see the in-depth explanation in Börner, Trilcke 2023), then containerization can be used as an overarching and integrating versioning mechanism.[3]

In the subsequent chapters, we will work through this set of problems and our proposed solutions in the following way:

[1] Cf. CLS INFRA deliverables 5.1 “Review of the Data Landscape”, https://doi.org/10.5281/zenodo.6861022 (Mrugalski et al. 2022), and 6.1 “Inventory of existing data sources and formats”, https://doi.org/10.5281/zenodo.7520287 (Ďurčo et al. 2022).
[2] In some sense, this dynamic and instability of the epistemic object is only part of the overarching dynamic of the CLS digital ecosystem. We have published some reflections on "CLS Research in Digital Ecosystems between Embeddedness and Instability" in our report “D7.1 On Programmable Corpora” (Börner, Trilcke 2023: 11–13).
[3] For an approach from computer science that points in a similar direction, see the concept paper "Establishing the Research Data Management Container in NFDIxCS” (Al Laban et al. 2023).

Chapter 3 (“Versioning Living Corpora Using Git Commits”) will, on the one hand, introduce Git[4] (in its actual implementation in the online service GitHub[5]), a powerful tool for *distributed version control*, as a way of versioning living corpora; on the other hand, using the GitHub API and the example of the GerDraCor corpus, we will illustrate what kind of additional (technical) metadata about living corpora can be retrieved.

Following chapter 3, we provide an **excursus** (“An Algorithmic Archaeology of a Living Corpus: GerDraCor as a Dynamic Epistemic Object”) in which we illustrate how Git-based versioning and the metadata that is produced in the course of versioning can be used for what we call *corpus archaeology:* an approach to the epistemic objects of CLS that treats them as technical objects whose genesis itself can be investigated. Crucial for our argumentation is the excursus because it vividly demonstrates what it means for a corpus to be "living".

Version control using Git is a viable solution for the stabilization of living corpora, as long as they are "just" data. However, this is not yet a sufficient solution for the stabilization of programmable corpora, as these are in fact combinations of data and code, whereby both are in a reciprocal relationship of co-evolution. As a consequence, the independent stabilization by versioning of data on the one hand and code on the other may not be sufficient in this case. Therefore, in **chapter 4** (“Dockerizing DraCor, for Example. On Versioning Programmable Corpora”), we present an approach that versionizes an entire programmable corpus as a bundle, using a containerization mechanism that we call "Dockerizing DraCor".

In the chapter 4 in particular, we will again follow a prototyping approach, as we have already done in our report "D7.1 On Programmable Corpora" (cf. our explanation in Börner, Trilcke 2023: 9–11). This also implies that the technical solutions we propose have mostly already been implemented as prototypes in the development work accompanying this report. We indicate where these technical prototypes can be found at the relevant points in this report. The information in chapter 0 ("About this deliverable") also provides an overview of the accompanying technical prototypes. Overall, our work revolves around the overarching "DraCor" prototype of a programmable corpora ecosystem: “DraCor” is a multicomponent prototype that includes a number of homogenized corpora and several APIs, some of which are document-based and some of which are research-driven; in addition, the DraCor prototype includes exemplary microservices.

[4] Git was originally introduced in 2005 by Linus Torvalds in connection with the development of the Linux kernel. For the history of Git, see e.g. the article "A Git Origin History" by Zack Brown (2018); for an introduction to Git, see e.g. Chacon, Straub (2024). The documentation can be found at https://git-scm.com/docs.
[5] Cf. https://github.com/

3. Versioning Living Corpora Using Git Commit

In the following, we show the capabilities of Git for the versioning and change tracking of living corpora. We will do this by describing and analyzing the evolution of a GitHub repository that contains a DraCor corpus. While in this PDF version of this report we only document the analysis, there is a web-based version of this analysis that is executable and fully reproducible.[6] In this PDF version of the report, our analysis is conducted with the German Drama Corpus (GerDraCor), but the method used (and the code implemented in the web-based version) will be largely applicable to any other DraCor corpus.

3.1 How to Better Not Cite a Living Corpus. An Example From Current Research

In this first step, we will take a short and exemplary look at an actual CLS research project and how it deals with the living corpora of DraCor. Our aim is to show that the way DraCor is cited is insufficient to enable reproducibility of the research.

It has become quite common for research that use DraCor corpora to

1) cite the paper Fischer et al. 2019[7]

2) include the information on how many plays are in the corpus used.

Plays used as examples are mostly referenced by author and title (and not, what we would recommend, by their DraCor ID[8]). This can, for example be observed in the following quotations of a research paper that uses GerDraCor to develop and test a tool using machine learning methods to detect chiasmi in literary texts:

> "We perform two types of experiments. [...] In the second experiment we evaluate how well our model generalizes to texts from different authors not included in the training data. To this end we extract PoS tag inversions from the GerDraCor corpus (Fischer et al., 2019) [...]" **The training data set** (https://git.io/ChiasmusData) "consists of **four annotated texts by Friedrich Schiller** *Die Piccolomini*, *Wallensteins Lager*,

[6] https://github.com/dh-network/clsinfra-d73

[7] It is understandable that this paper is cited, as it is listed on the main page of DraCor as a citation recommendation, as well as in the README file in the Github repository of GerDraCor. This is not intended as a criticism of the researchers who use DraCor resources, but rather as food for thought for us as creators of DraCor asking how users can be given a citation recommendation that actively promotes repeatability of research. This applies to information on the website, auxiliary files in the data repositories as well as in the design of the API responses: currently even information returned by the API, i.e. the responses of the endpoints `/info`, `/corpora` and `/corpora/{corpusname}` do not include the state or version of the data currently ingested into the database and thus available via the API.

[8] In the report "On Programmable Corpora" this is the feature P2 play_id, see "Tab. 02: Play Features", e.g. ger000086 of the play Die Piccolomini by Friedrich Schiller. These identifiers can be resolved via the /id/{id} API endpoint or the resolver at https://dracor.org/id/{id}.

> *Wallensteins Tod* and *Wilhelm Tell*. We annotated the whole texts, finding 45 general chiasmi and 9 antimetaboles." (Schneider et al. 2021 :98; emphasis [bold] by us)

And further

> "[...] we evaluate the generalization performance of our chiasmus classifier trained on the four annotated Schiller dramas to other texts. The **first set of texts comprises seven other dramas by Friedrich Schiller** [...]. To see how well our method generalizes to different authors, we tested it on the remaining **493 documents from GerDraCor**." (Schneider et al. 2021: 99; emphasis [bold] by us)

Although the authors publish their tool and the derived dataset[9] as open source resp. open data, it is not self-evident which version of GerDraCor was used. The only information that may support the identification of the version is the information about the number of plays "504"[10] included in GerDraCor at the time of assembling the training and test data set based on the corpus (and, of course, bibliographic metadata of the study itself, such as the date of publication). But in fact, there might be more than one version with 504 plays.

Based on this information, it is therefore not clear what data was used exactly in the study. However, this would be a problem for reproduction of this research. But how the problem could be solved? In the next chapter, we will show that there is a quite simple and elegant solution: Git commit history.

3.2 Citing Git Commits as Corpus Versions. Introduction

In our report "On Programmable Corpora" we have already introduced GitHub as a "key infrastructural component" in developing the DraCor toolset as well as in curating and hosting DraCor corpora. Previously, we have also relied on GitHub to directly link into the codebase of the DraCor API and other components of our ecosystem when explaining its inner workings (Börner, Trilcke 2023). However, in this section, it is the platform GitHub itself that is in the focus of our attention when we try to demonstrate how to effectively deal with datasets that are constantly in flux. Because DraCor is using Git (and respectively GitHub) for publishing corpora the process of creating and maintaining a corpus is fully transparent and traceable. As we will show, this also opens up unrivaled possibilities for versioning and the corresponding referencing of living corpora.

[9] https://git.io/ChiasmusData / https://github.com/cvjena/chiasmus-annotations
[10] 4 (annotated plays) + 7 other Schiller plays + 493 remaining documents from GerDraCor = 504 plays overall

Unlike the repositories of DraCor software components (cf. the repository of the DraCor API) for which releases are published,[11] in the case of corpus repositories this feature is (currently) not used.[12] However, it is still possible to very precisely point to a single “version” (or “snapshot”) of the data set. This can be done by referring to an individual *commit*[13]. Because all editing operations are “recorded” or “logged” when committed, the commits can be used to reconstruct the state of a corpus of a given point in time. We can consider the commits the “implicit versions” of DraCor corpora.

The GUI of GitHub already provides powerful tools to dive into the commit history of a corpus data set. The history of the repository

https://github.com/dracor-org/gerdracor

can be easily reached from the landing page (see Fig. 1).

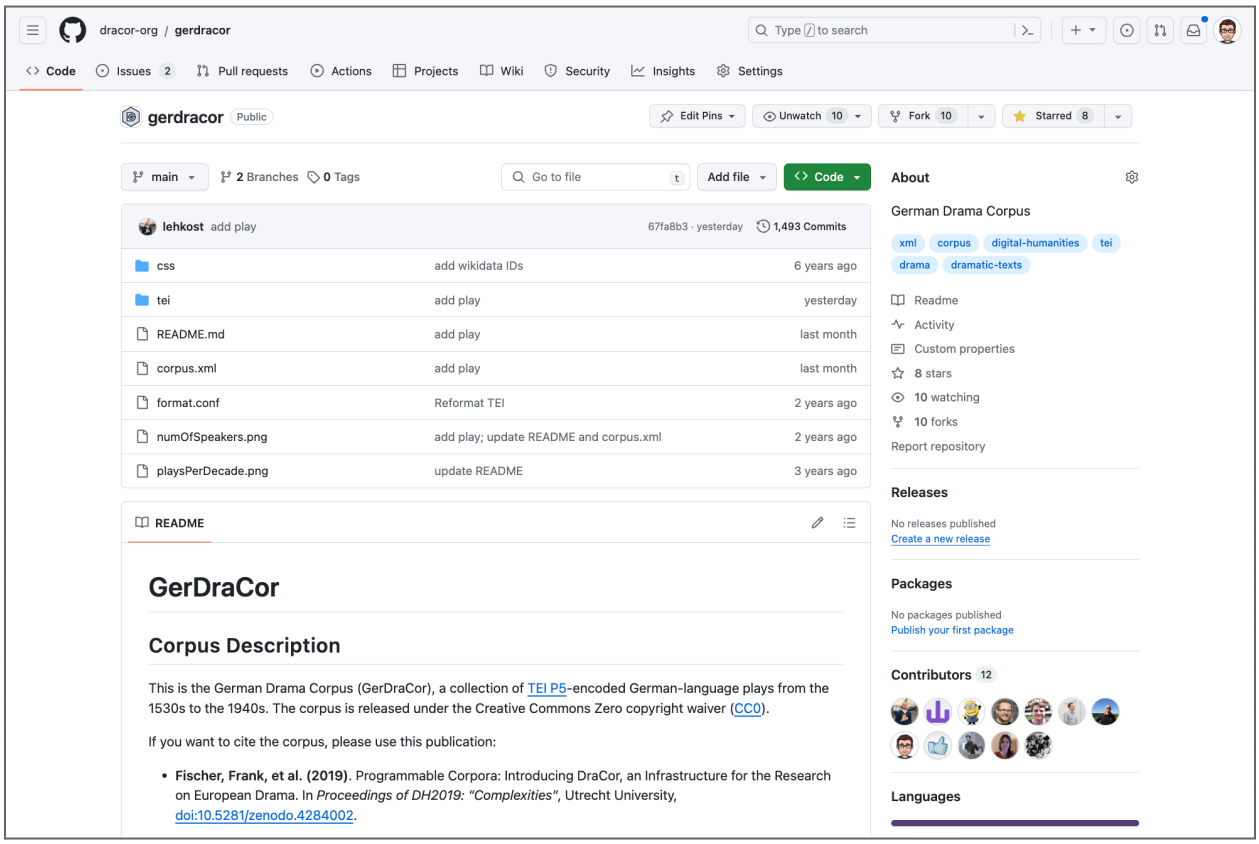

Fig. 1: Landing page of the repository of GerDraCor on GitHub

[11] https://github.com/dracor-org/dracor-api/releases
[12] On “releases” cf. the “GitHub Documentation” on https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases
[13] The glossary iof the Git documentation defines “commit” the following: “A single point in the Git history; the entire history of a project is represented as a set of interrelated commits. The word ‘commit‘ is often used by Git in the same places other revision control systems use the words ‘revision’ or ‘version’.” (https://git-scm.com/docs/gitglossary#Documentation/gitglossary.txt-aiddefcommitacommit).

The header above the file listing of the root folder (see Fig. 2) includes a link to the *latest commit*

https://github.com/dracor-org/gerdracor/commit/67fa8b39c90d4a1952d11c771b5d58175a8ccdf4

as well as the *commit history*:

https://github.com/dracor-org/gerdracor/commits/main/

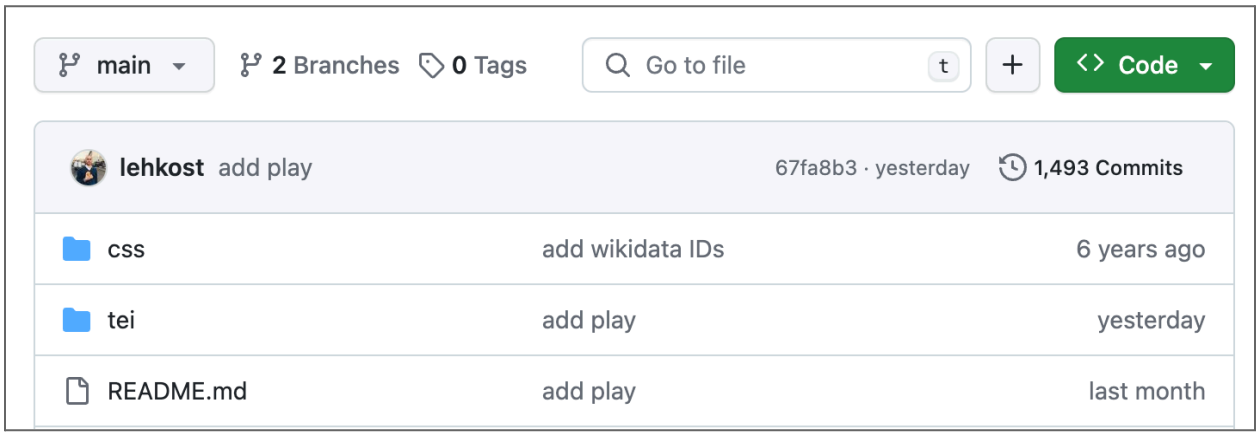

Fig. 2: Links to the most recent commit and commit history

The commit history allows for filtering commits by a certain date range, e.g. it is possible to display commits dating from February 2018:

https://github.com/dracor-org/gerdracor/commits/main/?since=2018-02-01&until=2018-02-28

From this list a single commit can be explored, e.g. from February 14th 2018:

https://github.com/dracor-org/gerdracor/commit/30760ec3ff4aa340f785bcc17bfd3ca81e7e2d06

This commit is identified by the SHA value (as "commit identifier") of "`30760ec3ff4aa340f785bcc17bfd3ca81e7e2d06`", which can also be found as part of the URL in the address bar of the browser.

From the single commit view it is possible to get to all TEI-XML files of the plays in the corpus at that point in time. This can either be done by clicking on the button "Browse files" in the upper right corner of the gray commit page header and then, on the landing page, by navigating to the folder `tei`; or, as a shortcut, by directly changing the URL in the address bar of the browser: To address the TEI files in the state of February 2018 the commit identifier `/tree/{commit SHA}/tei` can be appended to the URL of the GerDraCor repository `https://github.com/dracor-org/gerdracor`, resulting in:

https://github.com/dracor-org/gerdracor/tree/30760ec3ff4aa340f785bcc17bfd3ca81e7e2d06/tei

This example demonstrates that even without specialized tools and just by using the GitHub Web Interface it is straightforward to precisely retrieve a dated "version" of the corpus files. The only requirement is that the commit, or at least, the precise date or the date range in which the corpus was used is known.

3.3 Retrieving (Technical) Corpus Metadata via the GitHub API

To highlight the versioning capabilities of the Git commit history, we provide a detailed analysis of the commit history of a GitHub repository containing the data of a DraCor corpus. For this analysis, we developed a set of functions written in Python. The functionality of this prototype[14] of a tool is bundled as methods of the class `GitHubRepo` contained in the module `github_utils`[15].

To retrieve metadata about the commits and, thus, the state of a corpus (the "implicit version") at a given point in time, the GitHub API is used.[16] We will illustrate some functions of the API that are relevant in the following. Although we include URLs of concrete examples, the implemented methods to retrieve the data for the analysis in section 3.3 will work the same way.

A **list of commits of a repository including basic metadata** can be requested from the URL

https://api.github.com/repos/dracor-org/gerdracor/commits

This returns the commits in the repository in batches of 30 commits starting with the most recent one. The respective API operation is used to retrieve **the identifiers of the commits** (dictionary key SHA) and the **dates when the changes were committed.** We show the first lines of the returned JSON object:

```
{'sha': '67fa8b39c90d4a1952d11c771b5d58175a8ccdf4',
 'node_id': 'C_kwDOBH09MdoAKDY3ZmE4YjM5YzkwZDRhMTk1MmQxMWM3NzFiNWQ1ODE3NWE4Y2NkZjQ',
 'commit': {'author': {'name': 'Frank Fischer',
   'email': 'lehkost@users.noreply.github.com',
   'date': '2024-02-14T11:36:56Z'},
  'committer': {'name': 'Frank Fischer',
   'email': 'lehkost@users.noreply.github.com',
   'date': '2024-02-14T11:36:56Z'},
  'message': 'add play',
  …
 }
```

More detailed information on a single commit can be retrieved by attaching the SHA value to the URL of the commits endpoint. So, the detailed metadata of the commit identified by the SHA

[14] On our "reflective prototyping approach" cf. Börner/Trilcke 2023: 9ff.
[15] https://github.com/ingoboerner/d73/blob/main/report/github_utils.py
[16] See the "GitHub REST API documentation" on https://docs.github.com/en/rest?apiVersion=2022-11-28

"67fa8b39c90d4a1952d11c771b5d58175a8ccdf4" can be retrieved by sending a request to the URL

https://api.github.com/repos/dracor-org/gerdracor/commits/67fa8b39c90d4a1952d11c771b5d58175a8ccdf4

On the basis of the returned data it is possible, for example, to find out **which files had been added, modified, renamed or deleted** (see `status` in the `files` section of the response object) in a given commit. In the case of the commit in question, in the `files` part of the returned JSON object, the TEI-XML file "kotzebue-das-posthaus-in-treuenbrietzen.xml" of the play "Das Posthaus in Treuenbrietzen" by the author August von Kotzebue is listed with "added" as its `status` field value:

```
{'sha': '0f0008dfcb846ae837b0b5de55753ced5059f2cb',
  'filename': 'tei/kotzebue-das-posthaus-in-treuenbrietzen.xml',
  'status': 'added',
  'additions': 2592,
  'deletions': 0,
  'changes': 2592,
…
}
```

Another bit of information that is helpful when trying to reconstruct the state of a corpus, especially the files included, at a given point in time is **the "tree"[17] of the commit**. The respective URL to request this information is included in the basic commit metadata as well as in the more detailed response in the `tree` section.

```
{
…
'tree': {'sha': '3cbc81976a06a565d3ca673e3c17527bf6e30f8b',
   'url':
'https://api.github.com/repos/dracor-org/gerdracor/git/trees/3cbc81976a06a565d3ca673e3c17527bf6e30f8b'}
…
}
```

So the tree of the above mentioned commit can be retrieved at

https://api.github.com/repos/dracor-org/gerdracor/git/trees/3cbc81976a06a565d3ca673e3c17527bf6e30f8b

To access the **metadata of the individual files** containing the play data, the data folder has to be identified. As usual for DraCor, also in the case inspected here it is the `tei` folder:

```
{'sha': '3cbc81976a06a565d3ca673e3c17527bf6e30f8b',
```

[17] https://git-scm.com/docs/gitglossary#def_tree_object

```
 'url':
'https://api.github.com/repos/dracor-org/gerdracor/git/trees/3cbc81976a06a565d3ca673e3c175
27bf6e30f8b',
 'tree': [{
…
}
  {'path': 'tei',
   'mode': '040000',
   'type': 'tree',
   'sha': '64a98327331abbaa110fe9c9db11208aad3ced90',
   'url':
'https://api.github.com/repos/dracor-org/gerdracor/git/trees/64a98327331abbaa110fe9c9db112
08aad3ced90'}],
 'truncated': False}
```


So by requesting the data from

> https://api.github.com/repos/dracor-org/gerdracor/git/trees/64a98327331abbaa110fe9c9db11208aad3ced90

we receive information about the individual file objects, most notably the **filename** in the field with the key `path` and the **file size** (`size`) in bytes. See the following example:


```
{'path': 'achat-ein-april-scherz.xml',
 'mode': '100644',
 'type': 'blob',
 'sha': '87cd9f61a04cb322c0815afec694a49e4fd910b1',
 'size': 102669,
 'url':
'https://api.github.com/repos/dracor-org/gerdracor/git/blobs/87cd9f61a04cb322c0
815afec694a49e4fd910b1'}
```


Excursus: An Algorithmic Archaeology of a Living Corpus: GerDraCor as a Dynamic Epistemic Object

While we have shown in section 3.2 that Git commits can be used as stable references to states of living corpora and thus as a mechanism for transparent versioning, in section 3.3 we explored in more detail which information about corpora and corpus files can be queried using the GitHub API. Our explorations in section 3.3 have also demonstrated that the use of Git not only provides a powerful versioning tool for CLS corpora, but that this use of the API generates a large amount of additional (technical) metadata about corpora. In the following excursus, we use this metadata for a more in-depth analysis of the genesis of a DraCor corpus (namely “GerDraCor”), taking the Git commit history as a basis.

The aim of this analysis is twofold: On the one hand, we want to use a particular example to demonstrate what it means that in CLS we are occasionally dealing with living corpora. On the other hand, we want to use the analysis to illustrate the power of Git-based versioning and the metadata generated in this process, not least against the background that this might also open up new research perspectives on the epistemic objects of CLS (now understood as technical objects). An executable (and thus fully reproducible) version of this analysis is accessible on GitHub. In addition, we are convinced that an in-depth knowledge of the constitution of a data set is necessary when wanting to repeat the research based on this data. If the data has changed in the meantime between the original research and the repeating research knowing what exactly changed in the data can help to understand possible deviations in results and allows that informed counter-measures can be taken.

As already mentioned, the "German Drama Corpus" ("GerDraCor") will serve as our showcase. The corpus' repository is available at

https://github.com/dracor-org/gerdracor

The first step in the analysis is downloading all the data on all GerDraCor commits from the repository on GitHub. Depending on the overall number of commits this can take a long time. In a previous attempt, fetching and preparing the data of GerDraCor from GitHub took 53min 29s. The following analysis will be based on data that was downloaded on February 14th, 2024. At the time of the download the GitHub Repository "`gerdracor`" contained 1492 commits. We consider each commit being an implicit version of a corpus and therefore we have 1492 versions of "German Drama Corpus" up to this date.

The "Birth" of GerDraCor

From the commit history of the repository of the German Drama Corpus we can retrieve the very first commit. This initial commit to the repository which is identified by the SHA value "`2f4e830a852960eba8e05d6b622b3bd64911ab69`" was committed by Mathias Göbel (during a hackathon at the University of Potsdam) and dates from 2 December, 2016.

https://github.com/dracor-org/gerdracor/commit/2f4e830a852960eba8e05d6b622b3bd64911ab69

With this commit, 465 TEI-XML files were added to a data folder with the name `data`. The commit message "inital commit: converted text based on LINA and TextGrid" already reveals the initial source of the data of corpus: *LINA* is short for "Literary Network Analysis" and was the format developed in the project DLINA.[18]

[18] The project members included Frank Fischer, Mathias Göbel, Dario Kampkaspar, Christopher Kittel and Peer Trilcke. The output of the project is well documented on the project's Blog (https://dlina.github.io/), the data and tools are available on GitHub at dlina (https://github.com/dlina). The term LINA as an abbreviation of the German "Literarische Netzwerkanalyse" (Literary Network Analysis) first appeared in print in the publication Trilcke 2013.

In the DLINA project, research on dramatic texts was based on derivatives of the full-texts taken from the TextGrid Repository[19]. The “LINA files” included only metadata on the dramatic texts (e.g. date of publication) and very detailed structural information on the segmentation (acts, scenes); this included also information on which characters appear in which structural segment. Thus, the “Zwischenformat”[20]-Files (i.e., files in an intermediary format) allowed for the extraction of networks based on the co-occurence of characters in the same structural segment (cf. Dario Kampkaspar, Trilcke 2015). The DLINA project officially released their corpus of 465 plays before the Digital Humanities DH2015 conference in Sydney. This corpus is also referred to as “Sydney Corpus” or the corpus with the “Codename Sydney”. Unlike the current practice in DraCor, DLINA used Git Tags[21] on corpus data. In the GitHub Repository of the DLINA Corpus, there is a single tagged version `15.07-sydney`:

> https://github.com/dlina/project/releases/tag/15.07-sydney

The respective commit

> https://github.com/dlina/project/commit/cca01b501a1a294772c2a6a9fe38944b930eea03

adds a RelaxNG schema[22] describing the “Zwischenformat”. Although this dates from August 31, 2016, this version number was already introduced more than a year before in a blog post dating from June 20, 2015. The blog post explains it as such:

> “The version number 15.07 is referring to ‘July 2015’ as we’re going to present our results at the DH2015 conference on July 2, 2015. Further versions of the DLINA Corpus will receive according versioning numbers.” (Fischer, Trilcke 2015)

The files in the first commit to the GerDraCor repository re-include the full text of the play from the TextGrid Repository source in the element `<text>`, but in the `<teiHeader>` contain the metadata of the LINA files. This metadata is transformed from the custom intermediary format (“Zwischenformat”) that was used in the DLINA project to an XML encoding following the Guidelines of the Text Encoding Initiative (TEI Guidelines).

A Living Corpus “Growing”

Fig. 3 shows the development of the number of plays included in the corpus versions of GerDraCor. It can be seen from the plot that in 2018 the corpus slowly begins to grow.

[19] https://textgridrep.org
[20] https://dlina.github.io/Introducing-Our-Zwischenformat
[21] https://git-scm.com/docs/gitglossary#Documentation/gitglossary.txt-aiddeftagatag, see also https://git-scm.com/book/en/v2/Git-Basics-Tagging
[22] RELAX NG Specification. Ed. by James Clark, Murata Makoto. 3 December 2001. OASIS Committee Specification. https://www.oasis-open.org/committees/relax-ng/spec-20011203.html.

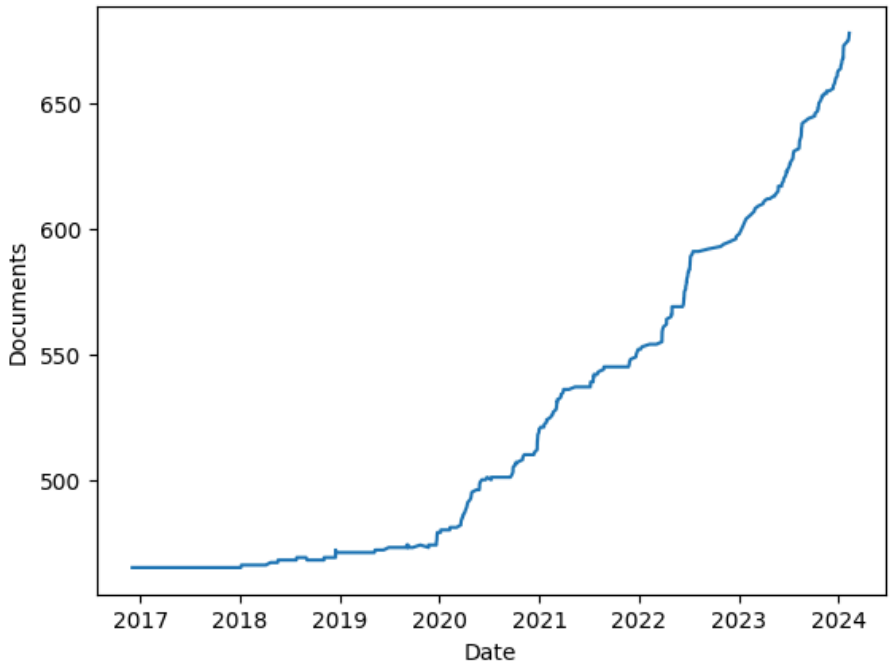

Fig. 3 Development of the number of documents in all versions in GerDraCor

Tab. 1 lists the number of plays being added per year from 2016 to 2024 which shows that from 2020 (the year the COVID19 pandemic began) onwards the number of plays added per year grows significantly.

year	new	overall
2016	465	465
2017	0	465
2018	7	472
2019	7	479
2020	41	520
2021	32	552
2022	45	597
2023	66	663
2024	15	678

Tab. 1: Plays added to GerDraCor per year

The “growth” of the corpus – to a certain extent – also results from diversifying the digital sources of which data is included.

Soon after the consolidation of the DLINA/TextGrid data the first Non-DLINA play was added. Corpus version “a0bf290a517092e3db27b5f37c30776f596565cd” dating from January 6, 2018 includes the play *Die Überschwemmung* by Franz Philipp Adolph Schouwärt[23]. The play’s digital source data does not stem from the TextGrid Repository as in the case of the data from the DLINA project[24], but is converted from Wikisource[25]. The following plot Fig. 4 shows the distribution of sources used over time.

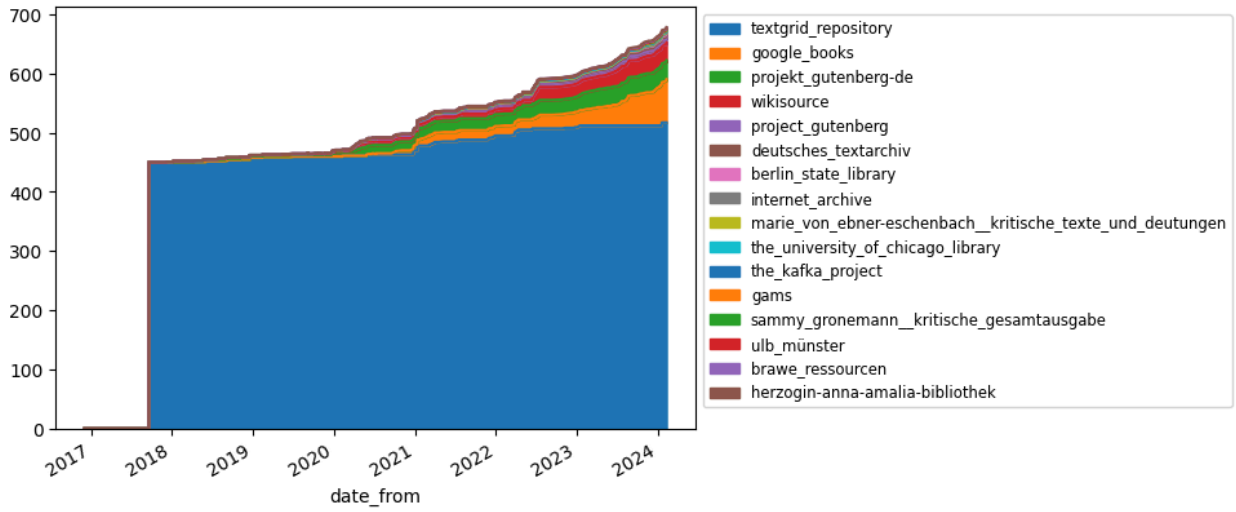

Fig. 4: Development of the Distribution of the Digital Sources of GerDraCor

In the course of its existence, the GerDraCor corpus also expands in terms of the time period covered (see Fig. 5).

[23] The identifier playname of the play is `schouwaert-die-ueberschwemmung`. The play can be accessed on the production instance of DraCor at https://dracor.org/id/ger000466.
[24] In a blog post (https://dlina.github.io/A-Not-So-Simple-Question) Fischer and Göbel [2015] conducted an evaluation of the data set “Digitale Bibliothek” (https://www.textgrid.de/Digitale-Bibliothek) that is included in TextGrid with the aim of identifying all available plays. The extracted TEI files can be found in this repository GitHub repository:
https://github.com/DLiNa/project/tree/master/data/textgrid-repository-dramas
[25] https://de.wikisource.org/wiki/Schouw%C3%A4rt_%E2%80%93_Die_Ueberschwemmung_(1784)

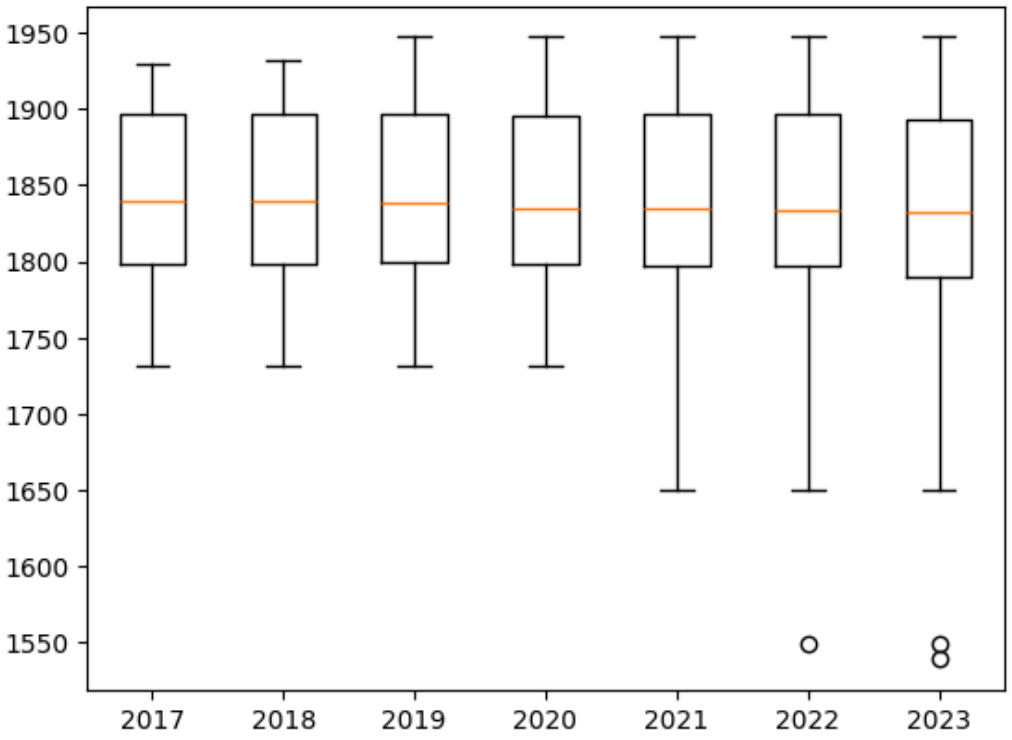

Fig. 5: Development of the time range ("YearNormalized") covered by GerDraCor

There is a second parameter that can be used to visualize the "growth" of a corpus – the sum of the file sizes of all TEI-XML documents in a corpus version. As expected, the overall size grows when adding plays to the corpus, but in 2017 it can also be observed that the overall size shrinks even though the number of plays stay the same. Still, plotting the size can be used to visualize changes when the overall number of documents stays the same, as happened in 2017, for example, when the total file sizes suddenly dropped. In this year no new plays were added to the corpus (see Tab. 1).

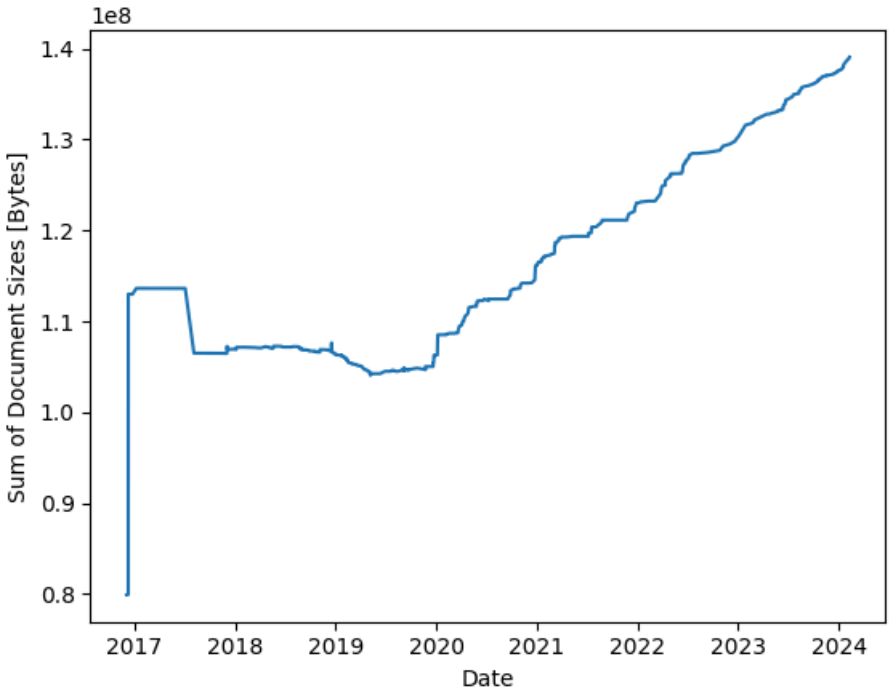

Fig. 6: Development of the sum of all document sizes in all versions in GerDraCor

We don't want to go into detail about what happened in 2017. What is more important to us at this moment is something else that clarifies our understanding of living corpora. Living corpora are not only characterized by the fact that the number of documents they contain is growing. Rather, it is also the case that the documents from living corpora themselves can change (e.g. grow or shrink), because they are enriched by additional mark-up or homogenized, which can lead, for example, to mark-up or metadata being deleted from the files (as was the case in 2017, to solve this cliffhanger).

Plotting the file size over a period of time is also useful to understand if and when a single file has been subject to modification. As an example we plot the file size of each version of the XML file of the play *Emilia Galotti* by Gotthold Ephraim Lessing[26] in Fig. 7.

[26] The play can be accessed on DraCor at https://dracor.org/id/ger000088.

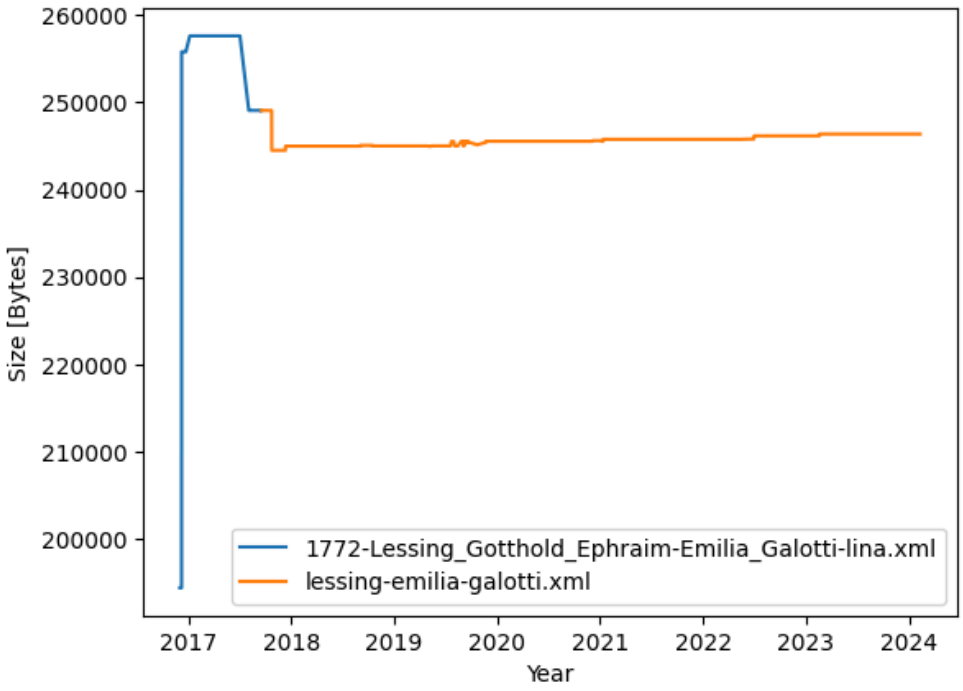

Fig. 7: Development of file size of the play Emila Galotti over all versions in GerDraCor

The change in color of the plotted line indicates the moment when the file was renamed from `1772-Lessing_Gotthold_Ephraim-Emilia_Galotti-lina.xml`[27] to the still valid `lessing-emilia-galotti.xml`. In the initial version of the XML file the metadata in the `<teiHeader>` was based on the LINA file, but not in the format ("Zwischenformat") that is available as `lina88` on the DLINA website[28], but already encoded following the TEI Guidelines. The reference to DLINA is still kept in the TEI version in the `<publicationStmt>` encoded as an `<idno>` element until the version dated '12 May, 2019' (see the section "Batch Edits as Major Revisions" below). A reference to the TextGrid source is contained in the `<sourceDesc>` and tagged as `<bibl type="digitalSource">` including a link to TextGrid in an `<idno>` element.[29]

The full text content taken from TextGrid was included in the initial version in the `<text>` element without any line breaks and indentations on a single line only. The text is formatted in

[27] See the file on GitHub: https://github.com/dracor-org/gerdracor/blob/2f4e830a852960eba8e05d6b622b3bd64911ab69/data/1772-Lessing_Gotthold_Ephraim-Emilia_Galotti-lina.xml
[28] https://dlina.github.io/linas/lina88/
[29] http://www.textgridrep.org/textgrid:rksp.0

the third version ("`106fefd275c20c5b673a326e89647f7511ba9f76`"[30]) which results in the steep increase of the file size. Overall, there are 26 versions in which the XML file of *Emilia Galotti* is modified.[31]

Batch Edits as Major Revisions

One can differentiate (at least) two major types of modifications to the files in a corpus version:

- Edits of single files only that bring a single file in shape;
- "Batch" edits that change all files at once.

The first edits need to be analyzed on an individual level. On a broader scale it must be said that these are, of course, relevant in reproduction studies because they might change, for example, the text resulting in different numbers of `<sp>`, stage direction, number of words, et cetera. We can always assume some effect on some of the metrics if there is a new version.[32]

In the following, we identify and comment on the batch edits, because we assume that they introduce changes to the files that are the results of some automatic process, e.g. using element Y instead of X.

In the case of the very early GerDraCor there are two commits that introduce significant changes to where files are stored in the repository and how they are named:[33]

---

30 https://github.com/dracor-org/gerdracor/commit/106fefd275c20c5b673a326e89647f7511ba9f76#diff-0d89c2bc1b36f1ca71e870eeb075d3f23d463ac76716a6709d7b7d8705adff6e

31 The the identifiers of the versions in which the file is modified are as follows: ['3e92970c6c3901b7661a515ce504605fb819e37d', '5d4130230eb9b9f2820618089ed1cf774424f241', '51dd69475f14757992c77d9a04c19026295b5f7c', '5b41d0f8f11fa478cfa68f9dac54912e6b922a63', '7987eb78ecee670e999373d1917bf64b8b1e5253', 'e8b7285eb4adbecebbcfcf53046f9a1093f25076', '879250ad0d9cc686dda930afa694d9461a0b5757', '8145e178ee1714ac115d2097e7c6df8cd1181e91', '1ab8c9b713a3eeaf5b25026e6036caaab230c119', 'e8c21cfd3b76bdcb05c6174e2c2d237d0d07c21e', '23833340aa83b963205f583189206a9881e9869e', 'd71e3d78562d8039f90bd21d88c6c7b67e912372', 'b0c6457ed5233ca634650d853627ed4ffe5fba8a', 'e1622273f41204a09d9af469d8f036a79b654af9', '1095d70934ab525f866a2f1978541eeedf618651', '8f0b2ac84f85ebf79df6b8aa0fb2d9662ee212ea', '9c2fcf55cce2ec6290ffc8615e98d9b2355707d9', '445a5b0fca0c96d7d4be7a9e8738a7c19cbe9cad', '7b4faf2f0d6e80a7e39052a38e12617453a1e5d2', '75b663876d1cd7b54235547ccd077a3299877a1c', 'fe53f8f5d4d36794df55859e262e6d1b893ce705', 'd23a93d9fa0e4eb53a580904ac5d01c8b8f8037c', '376cec4c609bc27cdcd9e2bb41bda7253c0174ff', 'bfadf6b5844d4e05ea0501898a23c21f71c10cb3', 'f7af1eb1060e94a916b856596d9e2e198f7159ea', 'a99060f0065856f8df114ce8556c31161c0332d1'].

32 There might be commits/versions in which only non-(play)-document files changed; usually these are unproblematic because these modifications do not affect the metrics and data returned by the API or any parts of files that are normally subject to an analysis, like the full text.

33 With the current setup of the the methods provided by our analysis tool `github_utils` (see section 3.3) it is necessary to take changes in file names into consideration if we want to investigate or visualize the changes of a file over the whole period. After the batch renaming in September 2017 the data folder name `tei` stays stable, but there is still some renaming happening. The method `get_ids_of_corpus_versions_renaming_documents` allows to retrieve the idendifiers of all versions in which document files are renamed. The method `get_renamed_files` returns dictionaries containing the version number as well as the old and the new filename. The following plays have been renamed after the above mentioned batch renaming: ['goethe-faust-eine-tragoedie',

- With the commit “e18c322706417825229f1471b15bd6daaeaf3ab1” dating from September 17, 2017 the folder containing the files is renamed from the initial `data` to `tei`.
- With the commit “fdac66ba90c2c094012dc90395e952411d324e4c”, on the same day, the file names of all TEI-XML files are changed to now match the identifier `playname`.[34]

There are 11 versions in which all TEI files available at that time are modified at once. In the following we describe the relevant ones.

- The commit “7987eb78ecee670e999373d1917bf64b8b1e5253” dating from September 3, 2018 adds Wikidata identifiers to 468 of 468 TEI files available at that time. The identifier is included as an element `<idno>` with the value "`wikidata`" of the attribute `@type` in `<publicationStmt>`.
- The commit “e8b7285eb4adbecebbcfcf53046f9a1093f250762“ dating from October 16, 2018 changes the `<revisionDesc>` of 468 of 468 TEI files available at that time. This is a modification of all files that should not affect any metrics currently returned by the DraCor API.
- The commit “8145e178ee1714ac115d2097e7c6df8cd1181e91” dating from May 12, 2019 mainly changes several elements in the `<teiHeader>` of 472 of 472 TEI files available at that time.

With the initial version of GerDraCor the identifiers that were assigned to the TextGrid plays in the DLINA project were kept. These identifiers consisted of a running number that, in the case of the DLINA website, was prefixed with “`lina`”. In this version of the corpus these identifiers were replaced with DraCor Identifiers that include the corpus identifier, e.g. “`ger`” followed by a number with 6 digits. For example, in the case of the play *Emilia Galotti* the LINA identifier “88” or “`lina88`” as in the URL of the LINA on the DLINA Website becomes “`ger000088`”.

This implies that all the 465 Identifiers of the DLINA-based plays in GerDraCor can be “converted” into the DLINA identifiers by just stripping the leading zeros and, in the case of the URL of the DLINA website, pre-pending “`lina`” in the URL:

‘malss-die-jungfern-koechinnen’, ‘neuber-die-beschuetzte-schauspielkunst’, ‘neuber-die-verehrung-der-vollkommenheit’, ‘lortzing-der-wildschuetz’, ‘schlegel-alarcos’, ‘hauptmann-die-ratten’, ‘hauptmann-florian-geyer’, ‘hauptmann-gabriel-schillings-flucht’, ‘hauptmann-vor-sonnenaufgang’, ‘holz-jerschke-traumulus’, ‘holz-schlaf-die-familie-selicke’, ‘laufs-jacoby-pension-schoeller’, ‘seemann-dulk-die-waende’, ‘stephanie-der-schauspieldirektor’, ‘stephanie-die-entfuehrung-aus-dem-serail’, ‘stephanie-die-liebe-im-narrenhause’, ‘stephanie-doktor-und-apotheker’, ‘barlach-der-blaue-boll’].

[34] In the report “On Programmable Corpora” (Börner, Trilcke 2023) this is the feature “P3 `play_name`”, see “Tab. 02: Play Features” (Börner, Trilcke 2023: 38). This identifier is used, for example, when requesting information about the play from the API: https://dracor.org/api/v1/corpora/ger/plays/lessing-emilia-galotti; Documentation of this API endpoint see https://dracor.org/doc/api#/public/play-info.

https://dlina.github.io/linas/lina{XXX}

E.g. the DraCor identifier that is resolvable with the URL

https://dracor.org/id/ger000010

would result in

https://dlina.github.io/linas/lina10

Vice versa

https://dlina.github.io/linas/lina465

would result in

https://dracor.org/id/ger000465

which is resolved to

https://dracor.org/ger/immermann-das-gericht-von-st-petersburg

The remaining major revision commits are:

- The commit “e8c21cfd3b76bdcb05c6174e2c2d237d0d07c21e” dating from September 7, 2019 assigns a RelaxNG schema derived from the DraCor TEI ODD[35] to 474 play TEI files available at that time.
- The commit “8f0b2ac84f85ebf79df6b8aa0fb2d9662ee212ea” dating from January 16, 2020 changes the values of the attribute `@xml:lang` of the root element `<TEI>` from ISO-639-1 to ISO-639-2 codes, e.g. from "de" to "ger" in all 480 play TEI files available at that time.
- The commit “9c2fcf55cce2ec6290ffc8615e98d9b2355707d9” dating from December 5, 2020 introduces the following modifications to all 510 TEI files available at that time:
    - date ranges, e.g. of the date of first publication of a play, are encoded using the attributes `@notBefore` and `@notAfter`, e.g. a previous encoded time span as in `<date type="written" when="1885">1884–1885</date>` becomes `<date type="written" notBefore="1884" notAfter="1885">1884–1885</date>`.

[35] ODD and schema can be found in the GitHub repository https://github.com/dracor-org/dracor-schema. When using the ODD derived schema to validate TEI-XML files of corpus versions of the past it would be necessary to also use the respective version of the schema, which would need to be retrieved from the commit history of the `schema` repository. It would make changes in the encoding more transparent if, at least for schemas or ODD, some sort of explicit version numbers would be used that could be then referenced in the TEI files of the plays. The TEI Guidelines define an attribute `@version` (https://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-TEI.html#tei_att.version) which “specifies *the version number of the TEI Guidelines* [our emphasis] against which this document is valid”. Strictly speaking, in the proposed use case the attribute would not point to an “official” TEI release but to a versioned ODD or `<schemaSpec>`element. This might be considered tag abuse and has still to be evaluated.

    - The diverse genre terms encoded as element `<term>` in `<keywords>` in the `<textClass>` section of the header that have been available in previous versions, e.g. `<term type="genreTitle">Schattenspiel</term>`, `<term type="genreTitle">Studentenspiel</term>` are either removed (resulting in `<term type="genreTitle"/>`) or normalized, e.g. the German `<term type="genreTitle">Komödie</term>` becomes `<term type="genreTitle">Comedy</term>`.
    - In addition to the normalized genre terms genre information is encoded using the element `<classCode>` in `<textClass>` linking to some selected concepts on Wikidata, e.g. `<classCode scheme="http://www.wikidata.org/entity/">Q40831</classCode>`. In this case the Wikidata identifier (Q-Number) identifies the concept “Comedy” (Q40831[36]) on Wikidata.[37]
- The commit “75b663876d1cd7b54235547ccd077a3299877a1c” dating from May 16, 2022 introduces a new element `<standOff>` to hold context metadata to all 569 TEI files available at that time.[38]
    - The wikidata identifier, previously encoded as element `<idno>`, is included in `<standOff>` as an element `<link>` with the value "`wikidata`" of the `@type` attribute and linking to the corresponding entity on Wikidata by including the concept URI in the `@target` attribute.

[36] https://www.wikidata.org/wiki/Q40831

[37] For the discussion of this encoding strategy and its implementation in the API see also the corresponding issue in the DraCor API repository: https://github.com/dracor-org/dracor-api/issues/120 (opened 3 December, 2020) which exemplifies the interdependence of API code and TEI-Encoding of the DraCor corpora. The class codes are the basis for the extractor mechanism which allows to include genre information in the API responses, see feature “P22 `play_genre_normalized`” in “Tab. 02: Play features” in the report “On Programmable Corpora” (Börner, Trilcke 2023: 38). Class codes that are supported are hard coded in the API code, e.g. https://github.com/dracor-org/dracor-api/blob/665ea3f07c3f0ab83566440436691ed73957b263/modules/config.xqm#L79-L89.

[38] The introduction of the `<standOff>` element is discussed in this issue in the DraCor schema repository: https://github.com/dracor-org/dracor-schema/issues/38 (the issue was opened on 14 March and closed on 2 July, 2022). This change of encoding is also a good example of the implications for other projects using DraCor corpora in their research: GerDraCor is used in several research projects, amongst them “QuaDramA” (https://quadrama.github.io/). In this project, two libraries were developed: “DramaNLP” (https://github.com/quadrama/DramaNLP) and “DramaAnalysis” (https://github.com/quadrama/DramaAnalysis). “DramaNLP” is the tool that generates the data to be analyzed with the R library “DramaAnalysis”. The changes introduced in the GerDraCor version discussed above had to be incorporated into DramaNLP as well, see the respective commit dating from 13 December, 2022 (https://github.com/quadrama/DramaNLP/issues/97).

    - The dates (printed, written, premiered) are also moved to the `<standOff>` container and are encoded as a `<listEvent>` with typed `<event>` elements.
- The commit “d23a93d9fa0e4eb53a580904ac5d01c8b8f8037c” dating from June 3, 2022 adds the DraCor ID as the value of the attribute `@xml:id` to the root element `<TEI>` and changes the encoding of the reference to Wikidata from a `<link>` to a `<relation>` with the value "`wikidata`" of the attribute `@type` to all 569 TEI files available at that time.

In this systematic perspective that is based on commits that change all files included in a corpus version at a time, we might still miss some important milestones in the development of the corpus if the respective changes in the encoding are not introduced at once but bit by bit to one or several files. For example, this is the case when introducing a new “feature”, the encoding of social or family relations of characters.[39] This feature was introduced to GerDraCor in cooperation with the “QuaDramA” project (Wiedmer, Pagel, Reiter 2020).[40] The elements `<relation>` in a `<listRelation type="personal">` in the `<teiHeader>` were added between September and November 2019 to 358 GerDraCor files.

What we have shown in this excursus was just a small example of the possibilities that a corpus archaeology can offer based on the information that the GitHub API makes available as the commit history. With the excursus, we wanted to illustrate how important it is from our perspective to familiarize oneself as comprehensively as possible with the epistemic objects of one's own research, i.e. –- in the case of CLS research — to gain a deep understanding of the origin, genesis and development of a corpus.

[39] Regarding the feature and its implementation in DraCor services, see the report “On Programmable Corpora” (Börner, Trilcke 2023: 48–49, 62).
[40] The pull request that included the data into DraCor dates from November 24, 2019: https://github.com/dracor-org/gerdracor/pull/10

4. Dockerizing DraCor, for Example. On Versioning Programmable Corpora

4.1 Containerizing a Research Environment

As already noted in the introduction, the use of a version control system like Git can be seen as both a powerful and user-friendly mechanism to stabilize “living” corpora. However, “programmable” corpora consist of a connection of data ("living corpus") with a lightweight research software (in the form of a research-driven API) whereby this connection can be conceptualized as a distributed research infrastructure in a dynamic digital ecosystem (cf. Börner, Trilcke 2023). Such a research infrastructure can no longer be stabilized only via the versioning mechanism of Git, since in addition to the two components (data, code), their specific connection is also crucial for the possibility of reproducing the research conducted with it. For such scenarios, as will be shown below, a container-based approach is suitable, whereby we will rely on containerization using the Docker software[41].

Our approach is inspired by the concept of research artifacts from computer science, which is described by Arvan et al. as “self-contained packages that contain everything needed to reproduce results reported in a paper" and which are also "typically self-executable, meaning that they are packaged within a virtual machine […] or within a container" (Arvan et al. 2022). The prototypical implementation, on which we will report in the following, is again based on the Drama Corpora Platform, DraCor. In a DraCor-based experiment, we set ourselves the challenge of making a network-analytic study, which we conducted for a paper publication, as fully replicable as possible using Docker.

The containerization technology “Docker" is widely used in the IT industry, because it can speed up development cycles and can reduce overhead when deploying applications. Especially in the field of “DevOps” – “Dev” for development and “Ops” for operations, which refers to the processes that are necessary to have an application run on a server – the importance of communication between the development and the operations team is considered highly important. Thus Docker workflows have been introduced. They do not only streamline communication processes, they also shift the responsibility of handling software dependencies to the development team: Docker enables the people actually writing the application to specify the environment in which their software should be run. There are a couple of key components to such workflows: The so-called “Dockerfile”[42] is a meaningful and executable form of

[41] https://www.docker.com

[42] “A Dockerfile is a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile.” (https://docs.docker.com/glossary/#dockerfile); For the documentation of the format see https://docs.docker.com/reference/dockerfile/

documentation. It contains the steps necessary to build a highly portable, self-contained digital artifact, a “Docker image”[43]. These images can be easily deployed on a designated infrastructure as “Docker containers”[44].

Certainly, in CLS research projects we will rarely find teams of development and operation professionals that are in need of communicating better. But, we would like to argue that still the attempt of reproducing research could be framed in a similar sense: On the one hand we have an individual researcher (or a team of researchers) that conducts a study. In our analogy, these are the developers. Their product – a study – relies on some application or script that operates on data. On the other hand we have researchers wanting to reproduce or verify the results, similar to operations professionals that have to deploy someone else's application on a server.

It becomes evident that some hurdles in the process of reproducing CLS research exist due to a lack of clear communication on how to run the analysis scripts and a tendency to offload the responsibility of setting up an environment in which the analysis could be executed to the reproducing party. A containerized research environment might circumvent these problems: Instead of claiming that scripts “work, at least as of today” on the machine of the developing researcher, as for example, Andrew Piper writes it in his foreword to his book “Enumerations” (Piper 2018: xii); when employing container technology, it could be guaranteed instead that a container created from an image containing a runnable self-contained research environment can be re-run. This would allow for a reproduction of the study. Instead of saying: The analysis was run-able on my machine when writing the paper and publishing the code only, in addition a researcher could provide a run-able research artifact alongside the study. When using Docker, this could be one or more images that would allow to re-create the research environment the study was conducted in.[45]

In the following, we will report on our exemplary experiment in making a CLS study replicable by using Docker technology.[46]

---

[43] “Docker images are the basis of containers. An image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.” (https://docs.docker.com/glossary/#image)

[44] https://docs.docker.com/glossary/#container

[45] On a side note and because someone might argue that by embracing a proprietary technology: “Linux containers” – what Docker is at its core – as a technology have been available since 2008. “Docker” was only introduced in 2013 and the company Docker Inc. actively promoted the “Open Container Initiative” (OCI, see https://opencontainers.org/), which was started in 2015. It developed a vendor agnostic specification of containers and images, that was released in 2017 to which the current Docker implementation adheres. This, of course, still doesn’t guarantee the longevity of the technology, but at least a total lock-in into a certain technology is circumvented.

[46] This experiment has also been presented at the “DH2023” conference in Graz. Cf. Börner, Trilcke et al. 2023. The slides of the presentation can be accessed at https://zenodo.org/records/8183676. For a comprehensive evaluation of the use of container technology in research and digital publishing see Burton et al. 2019

4.2 Case Study: Dockerizing a Complete CLS Study

We exemplify the benefits of a Docker-based research workflow by referring to our study "Detecting Small Worlds in a Corpus of Thousands of Theater Plays". In this study, we tested different operationalizations of the so-called "Small World" concept based on a multilingual "Very Big Drama Corpus" (VeBiDraCor) of almost 3,000 theater plays. As explained above, the corpora available on DraCor are "living corpora" – which means that both the number of text files contained and the information contained in the text files changes (e.g. with regard to metadata or mark-up). This poses an additional challenge for reproducing our study. Furthermore, our analysis script (written in R) retrieves metadata and network metrics from the REST API of the "programmable corpus". Thus, we had to devise a way of not only stabilizing the corpus but also the API.

DraCor provides Docker images for its services, which are, at its core the API, a frontend, a metrics service, that does the calculation of the network metrics of co-presence networks of the plays, and a triple store. The ready-made Docker images can be used to set up a local DraCor environment.[47]

For VeBiDraCor we devised a workflow that spins up a Docker container from a versioned bare Docker image of the DraCor database and ingests the data of the plays downloaded ("pulled") from specified GitHub commits using a Python script. We then committed this container with `docker commit`[48] to create a ready to use Docker image of the populated database and API. Because the build process is modular and documented in a Dockerfile, it is also possible to quickly change the API's base image or the composition of the corpus by editing a manifest file that controls which plays from which repositories at which state are included. In a second step, we also dockerized the research environment: a Docker container running RStudio to which we added our analysis script. The preparation of this image is documented in a Dockerfile. As base image we used an image or the rocker-project (https://rocker-project.org/). We used `docker commit` to "freeze" this state of our system and published all images. We call this state the "pre-analysis state", which is documented in a Docker Compose file.

After we ran the analysis, we again created an image of the RStudio container with `docker commit`, thus turning it into a Docker image in which basically "froze" the state of the research environment after the R-script was run. The image of this "post-analysis state" was

[47] Docker images of the DraCor system components are published on the platform DockerHub, see https://hub.docker.com/u/dracor. More recent images are dracor/api, dracor/frontend, dracor/metrics and dracor/fuseki. These are the images that are used in the production infrastructure and ideally should be used when setting up local instances as well. For reasons of allowing for replication of work that has, for example, been presented at the DH 2023 Graz conference, the image repositories with the naming convention that repeats "dracor" in the image name, e.g. dracor/dracor-api are still kept on the platform.
[48] https://docs.docker.com/reference/cli/docker/container/commit/

also published on the DockerHub repository. It allows for inspection and verification of the results of our study in the same environment that we used.[49]

By having these images representing two moments in the course of our analysis, we not only make our analysis transparent, we also allow for different scenarios of repeating our research. For example, starting an environment with the Docker Compose file that documents the "pre-analysis state" would allow a researcher to exactly repeat our analysis by re-running the script on the exact same data.

But also other scenarios of repeating research (e.g. replication, reproduction, revision, reanalysis, reinvestigation; cf. Schöch 2023) could be implemented easily. To give one example: A researcher could adapt the Jupyter Notebook we used to assemble "VeBiDraCor" and create an image of the local corpus container the same way we did. By changing a single line in the Docker Compose file documenting the "pre-analysis state" it is possible to start the whole system with this different data. For running our R-script to analyze this data he or she could still use a container created from our RStudio image and thus run the analysis the exact same way we did, but on different data.

4.3 Simplifying the Workflow: StableDraCor

The workflow presented above is still quite complex. There are multiple steps involved to set-up the locally running infrastructure some of which need to be run from the command line. In addition, one has to create a bare container with a database and an API, populate it with the data, conduct the analysis, create several Docker images, and, ultimately, publish them in a repository, e.g. on DockerHub. There is also the need to create the Docker Compose file that specifies which system components are needed to recreate the environment the analysis was run in at different points in time. Thus, there is a considerable need to make the process more user-friendly.

Our approach to simplifying the process focuses on developing a Python package called "StableDraCor" that makes the setting-up of local DraCor instances and populating them with data easier by somewhat "hiding" the complexity of the Docker and Docker Compose commands. While there is no real need for a generic tool managing containers and images (because this can be done with Docker Desktop[50]), with "StableDraCor" we address the complexity of setting up the specific DraCor infrastructure components and loading DraCor corpora (or a subset thereof).

---

[49] This process is documented in the README.md file in the repository accompanying the "Small-World" study, see https://github.com/dracor-org/small-world-paper/tree/publication-version. The original VeBiDraCor was built with a Jupyter notebook https://github.com/dracor-org/vebidracor/blob/3c3495d6b9434913687348435a341f781413304d/vebidracor-workflow.ipynb.

[50] https://www.docker.com/products/docker-desktop/

In the following, we describe the workflow using the tool. The package can be built and installed from the repository[51]. After initializing a StableDraCor instance, the infrastructure can be started with a single command `run()`. If a user does not specify any parameters (like pointing to a designated custom local Docker Compose file) the script fetches a configuration of the system specified by a Docker Compose file (`"compose.fullstack.yml"`) and starts the defined containers.

The package also supports setting-up local custom corpora either by copying a corpus or parts thereof from any running DraCor system, for example the production system on https://dracor.org or the staging server at https://staging.dracor.org (the latter containing even more corpora that are currently prepared for publication). It is also possible to directly add TEI files from the local filesystem, which allows a user to even use the DraCor environment with data not published on https://dracor.org or a public GitHub repository. When adding data to a local Docker container with the help of the "StableDraCor" package, the program keeps track of the constitution of the corpora and the sources used.

It is also possible to directly load corpora or parts thereof from a GitHub repository. This method of adding data allows to specify the "version" of the data in the corpus compilation process at a given point in time by referring to a single GitHub commit. As mentioned above, because DraCor corpora are "living corpora", it is not guaranteed that corpora that are available on the web platform do not change. Therefore, it would not be a good idea to base research aiming at being repeatable at the data in the live system. By using data directly from GitHub with StableDraCor it is possible to include only the plays that were available, let's say, two years ago and in the encoding state they were at this time.

The tool keeps track of the whole configuration of the system: This includes the versions of the microservices used, and the corpora loaded. The state of the corpus is identified by a timestamp and – if the source is a GitHub repository, the commit. This "log" can be output as a "manifest" JSON object (command: `get_manifest()`), which should allow re-creating the system even if no Docker image is available. It would also allow a user to unambiguously identify the exact data that was used in a study.[52] The following code snippet shows such a manifest with several corpora added:

```
{'version': 'v1',
 'system': {'id': '7f4f9ec9-40b2-4b92-8f33-5ef83714a12b',
  'name': 'my-stable-dracor',
```

[51] https://github.com/dracor-org/stabledracor. Currently, it is still recommended to use the version in development from the "dind" branch of https://github.com/ingoboerner/stable-dracor/tree/dind which implements a "Docker in Docker" setup. For a tutorial on how to use this setup see https://github.com/ingoboerner/stable-dracor/blob/dind/notebooks/02_intro.ipynb.

[52] If someone would not want to or for some reasons could not use Docker, the manifest alone would still be a sufficient source to retrieve the files used if they come from a corpus on GitHub. Of course, if local files are used, this does not help, but at least, it makes this circumstance transparent.

```
   'description': 'DraCor system created with the introduction notebook to showcase the
features of the stable-dracor-client.',
  'timestamp': '2023-11-23T13:25:31.445797'},
 'services': {'api': {'container': '8c9975f92468',
   'image': 'dracor/dracor-api:v0.90.1-local',
   'version': '0.90.1-2-g19a3f46-dirty',
   'existdb': '6.0.1'},
  'frontend': {'container': 'ac2e4e6d8a73',
   'image': 'dracor/dracor-frontend:v1.6.0-dirty'},
  'metrics': {'container': '3d8cc36cdf62',
   'image': 'dracor/dracor-metrics:v1.2.0'},
  'triplestore': {'container': '35802186a396',
   'image': 'dracor/dracor-fuseki:v1.0.0'}},
 'corpora': {'tat': {'corpusname': 'tat',
  'timestamp': '2023-11-23T13:25:32.435087',
  'sources': {'tat': {'type': 'api',
    'corpusname': 'tat',
    'url': 'https://dracor.org/api/corpora/tat',
    'timestamp': '2023-11-23T13:25:32.435093',
    'num_of_plays': 3}},
  'num_of_plays': 3},
 'dutch': {'corpusname': 'dutch',
  'timestamp': '2023-11-23T13:25:37.631544',
  'sources': {'dutch': {'type': 'api',
    'corpusname': 'dutch',
    'url': 'http://staging.dracor.org/api/corpora/dutch',
    'timestamp': '2023-11-23T13:25:37.631550',
    'num_of_plays': 1}},
  'num_of_plays': 1},
 'kar': {'corpusname': 'kar',
  'timestamp': '2023-11-23T13:25:41.428662',
  'sources': {'bash': {'type': 'api',
    'corpusname': 'bash',
    'url': 'https://dracor.org/api/corpora/bash',
    'timestamp': '2023-11-23T13:25:41.428668',
    'exclude': {'type': 'slug', 'ids': ['khudayberdin-aq-bilettar']},
    'num_of_plays': 2}},
  'num_of_plays': 2}}
```

[53]

"StableDraCor" supports creating a Docker image from a populated database container. With the original workflow it was necessary to do this with the `docker commit` command in the terminal. It was also necessary to provide additional documentation, for example the Jupyter notebook that was used to assemble a corpus. This information was 'detached' from the Docker image and it was necessary to explicitly point to this form of documentation, because it was not part of the research artifact itself. We tackled this issue with StableDracor and found a way to

[53] For an explanation of the manifest see https://github.com/ingoboerner/stable-dracor/blob/df31c4e6b42d0e8c6ba294efe4d26aa473719ab2/notebooks/02_intro.ipynb

include machine readable documentation *about* the research artifact directly *attached* to it. Now, when we create an image with the tool, we issue a slightly different “docker commit” command that also attaches Docker Object Labels[54] directly to the newly created image. We achieve this by taking the manifest as mentioned earlier, and decomposing it into single Docker Labels. StableDraCor can convert a manifest into labels but also re-convert Docker Object Labels on the image in the “`org.dracor.stable-dracor.*`” namespace back into a manifest. By providing the manifest information as image labels, we allow a user, for example, to retrieve information about the corpus contents and the sources of a database without having to run the image as a Docker container first. We also attach the information about the individual DraCor microservices directly to the image as labels. -

In summary, our “StableDraCor” package allows to generate a fully self-describing, completely versionized research artifact that alone is sufficient to replicate the corpora and their research infrastructure that were used in a study.

[54] https://docs.docker.com/config/labels-custom-metadata/

5. Key Takeaways

With this report, we first of all wanted to raise awareness of the challenges that research with living and programmable corpora entails if it claims to be reproducible research. Based on an exemplary corpus archaeology of GerDraCor, we have also illustrated the developments that a living corpus can undergo and that these developments can involve not only the number of corpus elements, but also the structure and metadata of the text files.

As we have shown, the problem of reproducing research on living corpora, i.e. dynamic epistemic objects, can be understood as a versioning problem. Accordingly, in this report we have introduced versioning techniques for living and programmable corpora and illustrated their use with examples taken from the DraCor platform.

To summarize, our recommendations on versioning are as follows:

- When working with living corpora, a version control system should always be used; when citing such corpora, the versions should be explicitly and comprehensively indicated. We recommend working with Git and citing Git commits accordingly (see chapter 3).
- When working with programmable corpora, in which data and code are linked and developed in dependence with each other, the entire infrastructure must be versioned. To this end, we recommend using container technology to conserve the state of data and code at the time when conducting the research. We recommend Docker for containerization (see chapter 4).

Beyond this, it is crucial for reproducible research that there is an overarching awareness and willingness among all those involved in the research process. We hope that this report will also help to strengthen this awareness.

References

Al Laban, Firas, Jan Bernoth, Michael Goedicke, Ulrike Lucke, Michael Striewe, Philipp Wieder, Ramin Yahyapour. „Establishing the Research Data Management Container in NFDIxCS“. *Proceedings of the Conference on Research Data Infrastructure* 1 (7 September 2023). https://doi.org/10.52825/cordi.v1i.395.

Arvan, Mohammad, Luís Pina, Natalie Parde. „Reproducibility in Computational Linguistics: Is Source Code Enough?“ In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, ed. by Yoav Goldberg, Zornitsa Kozareva, Yue Zhang, 2350–61. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, 2022. https://doi.org/10.18653/v1/2022.emnlp-main.150.

Börner, Ingo, und Peer Trilcke. „CLS INFRA D7.1 On Programmable Corpora“, 28 February 2023. https://doi.org/10.5281/ZENODO.7664964.

Börner, Ingo, Peer Trilcke, Carsten Milling, Frank Fischer, Henny Sluyter-Gäthje. "Dockerizing DraCor – A Container-based Approach to Reproducibility in Computational Literary Studies" In: *DH2023. Conference Abstracts*. https://doi.org/10.5281/zenodo.8107836

Brown, Zack. „A Git Origin Story“. *Linux Journal*, 27. Juli 2018. https://www.linuxjournal.com/content/git-origin-story.

Burton, Matt, Matthew J. Lavin, Jessica Otis, Scott B. Weingart. „Digits: Two Reports on New Units of Scholarly Publication“. *The Journal of Electronic Publishing* 22, Nr. 1 (14. Februar 2020). https://doi.org/10.3998/3336451.0022.105.

Chacon, Scott, und Ben Straub. *Pro Git: Everything You Need to Know about Git*. 2. ed. The Expert's Voice. New York, NY: Apress/Springer, 2014.

Da, Nan Z. „The Computational Case against Computational Literary Studies“. *Critical Inquiry* 45, Nr. 3 (March 2019): 601–39. https://doi.org/10.1086/702594.

Ďurčo, Matej, Vera Maria Charvat, Ingo Börner, Michał Mrugalski, Carolin Odebrecht. „CLS INFRA D6.1 Inventory of Existing Data Sources and Formats“, 27 July 2022. https://doi.org/10.5281/ZENODO.7520287.

Fischer, Frank, Ingo Börner, Mathias Göbel, Angelika Hechtl, Christopher Kittel, Carsten Milling, Peer Trilcke. „Programmable Corpora: Introducing DraCor, an Infrastructure for the Research on European Drama“. In *DH2019: »Complexities«. 9–12 July 2019. Book of Abstracts*. Utrecht: Utrecht University, 2019. https://doi.org/10.5281/ZENODO.4284002.

Fischer, Frank, Mathias Göbel. „A (not so) simple question and a somewhat diabolic answer“. *DLINA Blog* (blog), 18 June 2015. https://dlina.github.io/A-Not-So-Simple-Question.

Fischer, Frank, Peer Trilcke. „Introducing DLINA Corpus 15.07 (Codename: Sydney)“. *DLINA Blog* (blog), 20 June 2015. https://dlina.github.io/Introducing-DLINA-Corpus-15-07-Codename-Sydney.

Gavin, Michael. *Literary mathematics: quantitative theory for textual studies*. Stanford text technologies. Stanford, California: Stanford University Press, 2023. http://www.gbv.de/dms/bowker/toc/9781503633902.pdf.

Kampkaspar, Dario, Frank Fischer, Peer Trilcke. „Introducing our ‚Zwischenformat'". *DLINA Blog* (blog), 21 June 2015. https://dlina.github.io/Introducing-Our-Zwischenformat.

Kampkaspar, Dario, Peer Trilcke. „Editing Rules". *DLINA Blog* (blog), 22 June 2015. https://dlina.github.io/Editing-Rules.

Mrugalski, Michał, Carolin Odebrecht, Vera Charvat, Ingo Börner, Matej Durco. „CLS INFRA D5.1. Review of the Data Landscape", 19 July 2022. https://doi.org/10.5281/ZENODO.6861022.

Open Science Collaboration. „Estimating the Reproducibility of Psychological Science". *Science* 349, Nr. 6251 (28 August 2015): aac4716. https://doi.org/10.1126/science.aac4716.

O'Sullivan, James. „The humanities have a 'reproducibility' problem". *Talking Humanities* (blog), 9 July 2019. https://talkinghumanities.blogs.sas.ac.uk/2019/07/09/the-humanities-have-a-reproducibility-problem.

Schneider, Felix, Björn Barz, Phillip Brandes, Sophie Marshall, Joachim Denzler. „Data-Driven Detection of General Chiasmi Using Lexical and Semantic Features". In *Proceedings of the 5th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, ed. by Stefania Degaetano-Ortlieb, Anna Kazantseva, Nils Reiter, Stan Szpakowicz, 96–100. Punta Cana, Dominican Republic (online): Association for Computational Linguistics, 2021. https://doi.org/10.18653/v1/2021.latechclfl-1.11.

Schöch, Christof. „Repetitive Research: A Conceptual Space and Terminology of Replication, Reproduction, Revision, Reanalysis, Reinvestigation and Reuse in Digital Humanities". *International Journal of Digital Humanities* 5, Nr. 2–3 (6 November 2023): 373–403. https://doi.org/10.1007/s42803-023-00073-y.

Shanahan, Daniel R. „A living document: reincarnating the research article". *Trials* 16, Nr. 1 (11 April 2015): 151. https://doi.org/10.1186/s13063-015-0666-5.

Trilcke, Peer. „Social Network Analysis (SNA) als Methode einer textempirischen Literaturwissenschaft". In *Empirie in der Literaturwissenschaft*, ed by Philip Ajouri, Katja Mellmann, Christoph Rauen, 201–47. Münster, 2013.

Trilcke, Peer, Evgeniya Ustinova. „Detecting Small Worlds in a Corpus of Thousands of Theater Plays. A DraCor Study in Comparative Literary Network Analysis". In *Computational Drama Analysis: Reflecting Methods and Interpretations*, ed. by Melanie Andresen, Nils Reiter. De Gruyter, [forthcoming].

Wiedmer, Nathalie, Janis Pagel, Nils Reiter. „Romeo, Freund des Mercutio: Semi-Automatische Extraktion von Beziehungen zwischen dramatischen Figuren", 194–200. In *DHd 2020 Spielräume: Digital Humanities zwischen Modellierung und Interpretation. Book of Abstracts*. Paderborn: Paderborn University, 2020. https://doi.org/10.5281/zenodo.3666690