Immutable yet evolving: ARCs for permanent sharing in the research data-time continuum

Christoph Garth, Jonas Lukasczyk, Timo Mühlhaus, Benedikt Venn, Jens Krüger, Kolja Glogowski, Cristina Martins Rodrigues and Dirk von Suchodoletz

Scientific research data is often viewed as monolithic and immutable – once created and processed, it is published in archives for transparency and reproducibility. In this paper, we argue that research data sets should by default be viewed as dynamic and evolving, and should be created, managed, and curated by processes that mirror the development of software rather than via a publication-focused approach. We propose *Annotated Research Contexts* (ARCs), a lightweight basis for such processes, and illustrate how ARCs will assist research data management in plant biology, within a framework developed by the NFDI consortium DataPLANT.

1 Introduction

Research data management has received a growing amount of attention as many scientific disciplines are increasingly data-driven. For example, the *FAIR* principles [1] state that data should be made available in a findable and accessible manner, i.e., in open, publicly archives, and be interoperable and reusable, i.e. published in non-proprietary formats and annotated with metadata that describes contents and provenance. Mounting adoption of FAIR requirements by funding agencies — in particular, for publicly funded research — has greatly benefited overall quality, reuse, and sharing of research data. As a consequence, a plethora of open formats for FAIR data publication were developed in the past decade, for example the *schema.org*-based *Research Object* format [14] specification that is generic and domain agnostic, among many others.

While the FAIR principles define desiderata for *published* research data, there is an implicit presumption that data is published exactly once, and immutable henceforth. In practice, major incentives to publish or share research data are typically coupled to research cycles [2], e.g. sending a research manuscript for review to a publisher who requires

Das hier beschriebene Poster ist in der Open Access-Plattform der Universität Heidelberg heiDOK unter der DOI: `https://doi.org/10.11588/heidok.00029769` veröffentlicht.

data co-publication, or publishing all collected data at the end of a project. Focusing, or limiting, research data management processes to these two outcomes forgoes a host of opportunities that arise from a more continuous, dynamic approach. We will, in the following, argue that typical processes found in software development align well to the goals of research data management (RDM), and thus could be adapted to and adopted in an RDM context. We outline the current discussion on and status of the efforts of the DataPLANT consortium [3], within the framework of the National Research Data Infrastructure, to provide the technical basis — in the form of *Annotated Research Contexts* (ARCs) — for implementing corresponding processes for the plant research community. Our approach focuses not only raw data gathered during experiments, but also considers analysis routines and derived data products within the context of research data management.

2 RDM and Software Development

Software development is a well-studied aspect of Computer Science, and exhibits a comprehensive set of best practices to structure and manage the process of developing software systems at all scales, from small-scale personal software to largest-scale, multi-entity projects. These guidelines have emerged over half a century of practice. Others have written at length and much more formally about this [4]; here, it is not our intention to provide a detailed description. However, we single out a few broad insights and considerations that map well to the requirements of RDM, and from this derive requirements for a potential technical basis for RDM processes.

Holistic life cycle. Software development follows the goal completing a software product: a problem is analyzed, a design is derived, and an implementation is developed and released. This process is often cyclic, i.e. following a first release, requirements are reanalyzed, new features are designed, etc. However, the ultimate goal of each iteration of the cycle, i.e., producing a release, is anticipated throughout all steps. This has led to the development of techniques such as *unit testing* and *continuous integration*, which ensure that software is syntactically and semantically correct throughout the development phase. This also ensures that an implementation can be released at any point in time, e.g., if additional testing is required, or on a regular schedule.

RDM in most cases also follows a cycle: from a specific research question, experiments are performed to gather data, which is subsequently analyzed; results are published, and new or modified research questions are identified, restarting the cycle [2]. While publication of the data typically occurs at least at the end of each cycle, it appears beneficial to anticipate the need for publication already in earlier stages of the process, namely during data gathering and analysis. To give an example, the FAIR principles require that data is correctly annotated with metadata, such that others may interpret and reuse it. While it is absolutely feasible to perform this annotation just in time for publication, it often is a substantial burden of work, and in practice not performed with ideal diligence, resulting in metadata of insufficient quality [5].

Borrowing from the software development playbook, we advocate *continuous annotation*, i.e. spreading the data annotation throughout the RDM life cycle by ensuring that data is annotated as it is gathered, and annotations are carried through analysis. More generally, we argue that a data set should be annotated completely at every stage of its life cycle. The same applies to *continuous reproducibility*; instead of focusing a large effort on making analysis reproducible just before publication, analysis on a data set should always be reproducible.

At first glance, treating a large burden for a sequence of smaller burdens appears as a zero-sum game. However, in collaborative settings, the overall effort is quickly reduced as the number of collaborators grows. Moreover, ensuring that data is always well-annotated and reproducible affords opportunities for unplanned, ad hoc collaboration without incurring additional overhead.

Rapid and Scalable Collaboration A key aspect of software development is scalability: similar processes are used whether the development team is small or large. Effective collaboration is achieved using *distributed version control systems* such as e.g. Git, Mercurial, DARCS, etc., that store an evolving version of source code and arbitrate changes made by many developers.

It appears useful to use the collaboration opportunities afforded by version control also in the context of RDM [6], where collaboration also occurs on a variety of scales: from the individual researcher performing an experiment in isolation to a large multi-national inter-group collaborative effort, all forms of collaboration should be possible with minimal friction, while simultaneously ensuring that data is always in a consistent, well-annotated and reproducible state. Version control as a documentation and arbitration mechanism is ideally suited to this purpose, as it ensures an atomic and unambiguous history, without requiring diligence of participating researchers. Therefore, our goal is to amend ad hoc collaboration mechanisms (file sharing, email, etc.) by a systematic approach rooted in decentralized version control [7].

Automation A further ingredient in software development is that all aspects of development that can be automated, are automated. Two prominent examples are quality control (via continuous integration testing, see above) and adherence to common conventions (e.g., code formatting). RDM can borrow from this regarding a multitude of aspects. For example, the quality of metadata and reproducibility can be automatically assessed to a certain degree [8], allowing to quickly bring deficiencies to the attention of researchers and ensure continuous annotation and continuous reproducibility. Using these mechanisms, reviewers can rely on automation to ensure that a data set is in an advertised state. Finally, automatic annotation, where possible (e.g., for data products from computational analysis) can free researchers from manual annotation.

Provenance Often, software development requires a detailed understanding of the provenance of a particular aspect of an implementation. The history afforded by version control makes this straightforward. In the context of RDM, the detailed recording of changes through version control history, identifying the researcher responsible for each change, allows an in-depth understanding of who contributed in which manner to a data set, and in what form. While this information is crucial to collect in the first place, e.g. to be able to assign credit to all contributors to a data set, version control stands to effectively automate this previously tedious and manual process.

Cathedral vs. Bazaar In his seminal and highly influential essay, Raymonds [9] discusses two models for releasing source code during development, which are similarly applicable to the sharing of research data. In the *Cathedral* model, corresponding to a top-down approach, data would be made available at discrete releases, for example upon publication of a paper that relies on it. In contrast, in the *Bazaar* model, in a bottom-up manner, research data is constantly shared and evolved in full view of the public. His central thesis in contrasting these two models, adapted to an RDM context, is that data that is permanently shared during its evolution in a bazaar-type approach can benefit strongly from increased opportunities for collaboration and rapid identification of problems, such as lacking reproducibility or metadata annotation. Again, a version control approach to RDM may improve permanent sharing of research data and act as a catalyst toward increased data quality.

Open Standards It has been long recognized that an adherence to open standards, tools, and community-specific conventions is highly beneficial towards encouraging wide and opportunistic collaboration. A core consideration to support RDM technically is hence to rely on open and established tools with a wide user base, to the largest possible extent. For the case of version control, for example, this suggests to use the widely-used Git. Relying on overly specific or restricted solutions stands to exclude (opportunistic) contributions from others ("walled garden").

3 Annotated Research Context

A core objective in DataPLANT is to provide a technical basis that implements software development-inspired processes and integrates these into RDM workflows. While many technical solutions exist, the barrier of entry is significant due to their substantial complexity (e.g., version control systems); furthermore, the processes themselves must be negotiated, understood, and applied with stringency. This places a substantial burden on researchers and is a prohibitive time investment. Therefore, an additional objective of DataPLANT is to define and standardize easy-to-use RDM procedures and their technical realization, specifically targeted at the needs of the fundamental plant research community. In the following, we describe the *Annotated Research Context* (ARC) that DataPLANT is developing toward these goals.

ARC structure An ARC is a collection of files and folders, laid out in a specific schema, following the ISA model [10], which distinguishes *investigation*, *study*, and *assay* and is ubiquitous in the plant research community. ARCs are intended to capture all research data pertaining to a single study within a larger investigation; the scope of a single ARC is intended to scale from encompassing the research data of smaller research projects such as a single publication or a thesis, to larger lab-wide or even multi-lab investigations. We reserve a more detailed description of the ARC file structure, metadata schema, and workflow description for future work, once an initial specification is finalized.

ARCs are Version Control Repositories While based on a specific file structure, ARCs are version controlled repositories [6] and capture their evolution in the form of a version history. Version control system operations are not exposed directly to users, as their complexity is overwhelming to most non-experts and full flexibility is not needed. Rather, we define an **update** operation that records the current state of all files in the history. In addition to allowing an understanding of the provenance of an ARC, update operations are canonical points for automation of quality control and other tasks.

Git is a suitable initial choice for the underlying version control system, due to several factors. It is very lightweight and decentralized, allowing researchers to operate individual ARCs without overhead or requiring a centralized resource. Furthermore, Git is technically mature, widely adopted, and very well documented. Others have employed Git for RDM with same reasoning (cf. Section 4).

Collaboration with ARCs ARCs can be easily used collaboratively shared through existing Git repository hosting mechanisms, such as e.g. Github or a Gitlab server instance. To keep complexity manageable, low-level Git operations *push* (propagating local changes to a remote repository) and *pull* (merging remote changes with the local history) are replaced by a **sync** operation which combines these two. If conflicts occur, i.e., when remote changes would overwrite local changes or vice versa, researchers can opt to overwrite either local changes or remote changes, or address the conflict manually.

Multiple researchers can collaborate on a single ARC by selecting a hosting facility (e.g. public or lab-specific) and placing a central copy of their ARC there. They can concurrently modify and **update** their respective ARC copies, and **sync** to propagate their own changes to the central ARC and obtain the other researchers' changes. Moreover, ARCs will support an **import** operation to selectively reuse parts of one ARC (e.g. a data set or workflow) in another, while retaining the capability of sending changes back to the original ARC.

Within DataPLANT, the **DataPLANT** hub will provide a central hosting facility for ARC Git repositories, open to all researchers participating in the consortium and their collaborators, further reducing the burden of initiating collaboration with others.

ARC Automation Towards ensuring continuous annotation and continuous reproducibility, the version control inherent in ARCs could serve as a vehicle to easily implement automation for each new version. Currently, automation is considered in checking for adherence to file schema, metadata completeness and quality, reproducibility, among other aspects.

ARC Sharing and Publication Publication of ARCs is fairly straightforward [11], especially using services that already allow formal publication of Git repositories, such as e.g. Zenodo.org or the Open Science Foundation (osf.io). The DataPLANT hub will also offer publication facilities. ARCs, with full modification history, will be archived long-term and accessible via DOI.

To avoid lock-in into a specific ecosystem of tools and ensure that ARCs play well in the diverse ecosystem of research data publication facilities, it is planned to offer automatic conversion of ARCs into other ubiquitous formats for research data archival, such as e.g., *RO Crates* [12] or formats compatible with OpenAIRE / H2020 Data Management Plans. Unless explicitly hosted on private repository servers, ARCs are shared by default, in real time — no additional steps are needed to share an ARC and its evolution with others.

Implementation. ARCs and the RDM processes they support are conceptually straightforward and rely on open standards and tools (e.g., Git). To support the RDM procedures outlined above without having to resort to low-level operations (e.g. *git push*) dedicated support software is currently under development, initially as a command line tool, and later as a GUI client and web-based interface. Several alternatives are currently under consideration.

4 Discussion & Conclusion

Naturally, others have pursued a similar approach to the one we propose here, and unsurprisingly come up with similar designs to support RDM in daily use [11]. We will here briefly consider two representative concepts (resp. tools) to illustrate similarities and differences to our approach.

The notion of fundamentally conducting RDM on the basis of version control has been investigated in a broad manner [6]. For example, the powerful and domain-agnostic *Datalad* tool [15] is based on a Git variant (git-annex) to manage and describe the evolution of research data and enable low-friction collaboration, and supports a variety of generic operations. However, this generality incurs complexity: clear processes are not defined. Defining and negotiating these between collaborators can be a substantial burden towards effective collaboration. In contrast, the ARC process trades generality for convenience, focusing on few important operations relevant to plant research workflows. For example, while ARCs are always self-contained, Datalad provides a more general form of linking to other data sets, which however makes it difficult to check data set completeness (e.g., for archiving) or provenance. The *DVC* tool [13] takes a more holistic view and adds tailored processed for Machine Learning to versioned data management. It is related to our approach, however, but its specificity prohibits an application in plant research.

ARCs are designed as a support vehicle for RDM processes in plant research, and borrow in design from software development best practices. The ARC concept at heart embodies the idea of permanent sharing of research data. Fundamentally, this implies a strong reliance on open standards and tools, and version control as a candidate supporting technology. In this manner, ARCs are a key driver towards fulfilling DataPLANT's mission: to enable plant researchers to participate as first-class citizens in a large, vibrant, and growing ecosystem of data-driven research.

Acknowledgements

We acknowledge support for DataPLANT 442077441 through the German National Research Data Initiative (NFDI 7/1).

ORCID IDs

- Christoph Garth
- Jonas Lukasczyk
- Timo Mühlhaus
- Benedikt Venn
- Jens Krüger
- Kolja Glogowski
- Cristina Martins Rodrigues
- Dirk von Suchodoletz

Bibliography

[1] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

[2] Philippa C Griffin, Jyoti Khadake, Kate S LeMay, Suzanna E Lewis, Sandra Orchard, Andrew Pask, Bernard Pope, Ute Roessner, Keith Russell, Torsten Seemann, et al. Best practice data life cycle approaches for the life sciences. *F1000Research*, 6, 2017.

[3] Dirk von Suchodoletz, Timo Mühlhaus, Jens Krüger, Björn Usadel, and Cristina Martins Rodrigues. Dataplant – ein nfdi-konsortium der pflanzen-grundlagenforschung. *Bausteine Forschungsdatenmanagement*, (2):46–56, 2021.

[4] David King and David Katch. *Current Practices in Software Development: A Guide to Successful Systems*. Yourdon Press Englewood Cliffs, New Jersey, 1984.

[5] Rafael S Gonçalves and Mark A Musen. The variable quality of metadata about biological samples used in biomedical experiments. *Scientific data*, 6(1):1–15, 2019.

[6] Christian T Jacobs and Alexandros Avdis. Git-rdm: A research data management plugin for the git version control system. *Journal of Open Source Software*, 1(2):29, 2016.

[7] Eric C Kansa, Sarah Whitcher Kansa, and Benjamin Arbuckle. Publishing and pushing: mixing models for communicating research data in archaeology. *International Journal of Digital Curation*, 9:57—70, 2014.

[8] Vidya Ayer, Christian Pietsch, Johanna Vompras, Jochen Schirrwagen, Cord Wiljes, Najko Jahn, and Philipp Cimiano. Conquaire: Towards an architecture supporting continuous quality control to ensure reproducibility of research. *D-Lib Magazine*, 23(1/2), 2017.

[9] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

[10] ISA TAB format. `https://isa-specs.readthedocs.io/en/latest/isatab.html`, accessed 2021-04-26.

[11] Jean-Baptiste Poline. From data sharing to data publishing [version 1; referees. 2018.

[12] Eoghan Ó Carragáin, Carole Goble, Peter Sefton, and Stian Soiland-Reyes. RO-Crate, a lightweight approach to Research Object data packaging, July 2019.

[13] DVC – open-source version control system for machine learning projects. `https://dvc.org`, accessed 2021-04-26.

[14] Eoghan Ó Carragáin, Carole Goble, Peter Sefton, and Stian Soiland-Reyes. A lightweight approach to research object data packaging. In *Bioinformatics Open Source Conference (BOSC) 2019*, 2019.

[15] Yaroslav O Halchenko, Benjamin Poldrack, and Michael Hanke. Datalad–decentralized data distribution for consumption and sharing of scientific datasets. In *Organization of Human Brain Mapping Poster. Organization of Human Brain Mapping Annual Meeting, Geneva, Switzerland*, 2016.