EPiC Series in Computing
Volume 95, 2023, Pages 229–238
Proceedings of European University Information Systems Congress 2023

Supporting Software Development Processes for Academia with GitLab

Marius Politze[1*], Uta Christoph[1], Bernd Decker[1], Petar Hristov[1†], Ilona Lang[1‡], Marcel Nellesen[1§] and M. Amin Yazdi[1**]
[1] RWTH Aachen University, Germany
politze@itc.rwth-aachen.de, christoph@itc.rwth-aachen.de, decker@itc.rwth-aachen.de, hristov@itc.rwth-aachen.de, lang@itc.rwth-aachen.de, nellesen@itc.rwth-aachen.de, yazdi@itc.rwth-aachen.de

Abstract

Over the past years, several software development teams in the IT Center have continuously improved their development processes. One of the most significant changes was to select GitLab as the central tool for the software-development-life-cycle and to enhance certain processes with the help of this tool in the areas of planning, implementation, interaction in the team and with stakeholders, operations, and deployment. Subsuming the shared knowledge of all these teams, this paper presents the resulting best practices with regards to previously established workflows from the viewpoint of projects supporting university IT services in the areas of student-life-cycle and research data management.

1 Introduction

Digitization is a constant driver in higher education and research. Universities' IT services, therefore, are constantly professionalizing their workflows. However, it is often hard to adapt new technologies and modern processes as existing "legacy" systems often need to be supported for extended periods of time. As a result, systems have grown over time and sometimes depend on proprietary solutions. Additionally, we often see a clear separation of concerns in different teams for operation of IT systems, development of custom solutions, and adaptation of existing, often commercial, products. This makes it especially hard to extensively adopt invasive technologies like

[*] https://orcid.org/0000-0003-3175-0659
[†] https://orcid.org/0000-0003-0527-9189
[‡] https://orcid.org/0000-0002-7202-5982
[§] https://orcid.org/0000-0002-1830-5780
[**] https://orcid.org/0000-0002-0628-4644

Kubernetes. However, general software-life-cycle best practices can be applied independently of the underlying technologies and can mostly be gradually incorporated into existing working practices. For this purpose, several departments at the IT Center of RWTH Aachen University use GitLab for their daily work. Therefore, this paper draws mainly from two areas of application: (1) for the development of applications that support researchers in managing their research projects and data according to the FAIR principles (Wilkinson, et al., 2016) and (2) for the development of applications that support processes in learning and teaching in addition to a student-life-cycle management system (Yazdi & Politze, 2020). In this paper, we present a combined list of experiences and best practices from several teams totaling about 35 developers and stakeholders ranging from students, university staff, and faculty members to externals in third-party-funded projects. Even though the experiences come from software development for academic administration and are highly opinionated, other fields like research data management (Cyra, Politze, & Timm, 2022; Hahn, et al., 2017; Halchenko, et al., 2021), research software engineering (Flemisch, Gläser, & Politze, 2020; Colomb, 2020) or education (Küppers, Politze, & Schroeder, 2017) see similar developments and allows sharing such best practices.

2 Cases

Different teams have adopted a wide range of features of the software-life-cycle platform GitLab to support their daily work. While some are more limited to the teams that do software implementation work, others can be reused widely in the context of academic digitization, like requirements engineering, stakeholder communication, documentation, and IT infrastructure management. Based on the different teams' shared experiences, the following sections give an overview of these in the form of best practices implemented in GitLab, where they support software development and adjacent processes.

2.1 On Planning

Form teams that work together and give them access to the information required. Forming groups and subgroups of people working together on projects is a feature that is native to many, if not all, project management platforms. In terms of permissions, GitLab stays quite simplistic in allowing five levels of roles "*Owner*", "*Maintainer*", "*Developer*", "*Reporter*", and "*Guest*". Where successive roles only have a strict subset of permissions of their predecessor. Group and subgroup relationships can have multiple levels but are strictly tree-like and hierarchical, with projects being the leaves. Permissions given on a higher group level can only be extended further down the tree of groups and never be reduced: E.g., a "Reporter" in the group has at least this level of permissions for all subgroups and projects but can be made a "Maintainer" in a subset. While this restriction can sometimes be limiting, the resulting sets of permission are very comprehensible for users.

In the academic context where we usually work with constantly changing personnel, e.g., due to limited contracts, student workers, or thesis works, the following rule of thumb for assigning roles has proven practical:

- *Owner*: Group or department leads, mainly responsible developers, and an overarching service account to ensure access.
- *Maintainer*: Senior developers responsible for the main development work.
- *Developer*: Junior developers, student workers, students, and other (non-developing) staff.
- *Reporter*: Stakeholders that need read access to the source code or (external) developers that should fork the source code to contribute, e.g., based on a Project Forking Workflow[1].
- *Guest*: Stakeholders that require access to read packages, containers or that contribute to Issues

[1] see https://docs.gitlab.com/ee/user/project/repository/forking_workflow.html

Especially *owner* roles can be implemented by following a policy that requires all projects to be created under one or a few centrally maintained groups. Hence, *owners* are hierarchically propagated to subgroups and projects.

The most prominent features within the projects that support project management are Epics on a group level and Issues within the individual projects. GitLab supports several ways of visualization for project planning, like Gantt or Board views, which is extremely common in agile software development methodologies like Scrum or Kanban. Thus, using Epics and Issues enables the implementation of Scrum or Kanban-based agile processes. In principle, Epics and Issues are widely similar but should be regarded hierarchically: An Epic can be comprised of multiple Issues.

While both, Epics and Issues support some specific metadata fields like a due date or assignees, in our cases, the most used features are a structured description and labels. Since description is a free text field, it is required to agree with the team on which information should be present in any Issue or Epic. For that, our teams that are focused on software development have created templates containing essential but required information such as:

- Description of the issue/task: A free text field to briefly describe a task or bug in a few sentences.
- Stakeholders / Personas: A note on the person that brought up the idea or that needs to be updated once it is finished or who profits from the completion of the task. For people working on the tasks, this might help to understand the context in which the Epic was created. In academia, personas might be by scientific and teaching faculty members, other (administrative staff), students, external project partners, the ministry of education, etc.
- Business outcome or value for users: A text briefly describing the aim or the goal of the stakeholders.
- Time criticality or constraints: Most business cases are bound to some time constraints. This is no different within academics, as deadlines are often bound to the end or beginning of a semester.
- Steps to reproduce: Mostly in case of an error report: a text briefly describing how a developer can reproduce the error to understand the technical details.
- Acceptance Criteria: Minimal technical and non-technical requirements that need to be fulfilled to complete the task.
- Definition of Done: A generic checklist of things to consider before a task is being regarded as done, e.g., inform stakeholders, update documentation, write a tweet about the task, conduct a four-eye principle, and clean up a database.

For less software-driven teams, the headlines may be partially or entirely different. Nevertheless, the team should agree on the required information and create a template for their Issues and Epics that suits the processes within their workgroup. In our case, the Epic template is based on the Epic hypothesis statement from the Scaled Agile Framework (SAFe)[2] (see below) and is adapted to the team and stakeholder-specific needs. Moreover, the templates are created so that every team member, developer, or other staff can create Issues and Epics and put in required context information so that any colleague commences working.

The description is by far the most extensive piece of information. However, of much more practical use are so-called labels. Labels offer attaching tiny bits of information to an Issue or Epic to sort them into certain categories. GitLab uses these labels for example to implement workflows like “Todo”, “In Progress”, and “Done”. However, labels can be used for all kinds of quantitative categories. For example, rating according to MoSCoW, ROI or criticality schemes, reporting information for funding or stakeholder involvements or required competencies to complete the task. While this feature seems simplistic it can be used to annotate Epics and Issues with various dimensions. Using the filtering and

[2] see https://www.scaledagileframework.com/

visualization capabilities of GitLab then allows one to get a quick overview of the status of the project (cf. Figure 1).

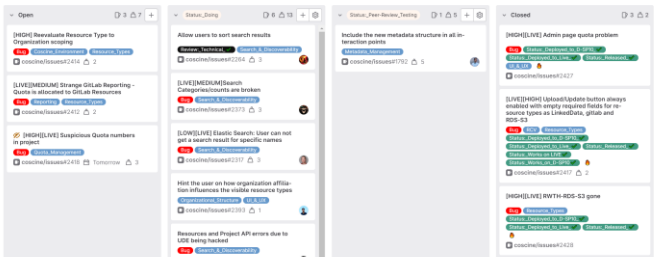
While smaller and larger day-to-day tasks can be tracked with the Issues and Epics, long-term documentation requires a less workflow driven and more persistent way of storage. Most projects use some form of Wiki for this purpose. GitLab offers two ways to approach this: either within the code repository (directly next to the code) or within a specified wiki repository. Both ways offer roughly the same set of functions. For more development-oriented teams, it has proven more practical and increased acceptance to store wiki information directly within the source code repository. In contrast, other teams have decided to disable the source code repository and use the separate wiki functionality. Both alternatives use Markdown syntax to structure the information and to allow some rich text rendering using headings, links, images, or flowchart-based drawings.

2.2 On Implementation

One core task for software development-oriented teams is the implementation of software. However, implementation workflows can and should be considered from a wider angle and for many other IT related tasks e.g., management of configuration files, databases, or home pages. Therefore, tasks like these should be considered implicitly included when referring to the development workflow below.

Introducing Feature Branches is probably one of the most intimidating changes to a development workflow when starting to work with Git (not even necessarily with GitLab). Feature branches allow multiple developments in parallel with little to no influence on each other and different development cycles depending on the "size" of the longevity of a feature. Feature branches are also the basis for many of the automation and communication features of GitLab, like CI, Merge Requests, or Environments. Various branching strategies exist; each project needs to pick one that fits its needs. The overall experience has shown that starting with simpler models and then extending them as needed has had a high success rate and adoption among the development team. Start using Git with a single "main" branch only. Add a "staging" or "development" branch depending on the envisioned release workflow. Integrate local changes early in the process. Once the workflow gets more complex add feature branches. All our teams have adopted branches for development ("dev") and stable ("main") states of their source codes. Teams following a stricter Scrum process also have widely adopted feature branches used to hold intermediary versions before being merged into a shared development branch during the sprint. Together with the planned tickets, the teams have mostly agreed on a three-stage-workflows "Develop", "Review", "Deploy" as shown in Figure 2.

Continuous integration (CI) pipelines are commonly used to support developers when testing or integrating their feature branches. While CI is a core feature within the GitLab UI, it requires an additional runner server, which can be operated on a single, shared virtual machine even for multiple teams. Within these pipelines, which consist of a series of jobs, the current state of the source code can be checked for certain types of errors like consistency or syntax. A series of automated unit or validation tests verify that the code does not re-introduce previously known bugs or violate specifications. While this pattern is long known within software development, the pattern can be re-used to validate


Open 3 7

[HIGH] Reevaluate Resource Type to Organization scoping
Bug Coscine_Environment Resource_Types
coscine/issues#2414 2

[LIVE][MEDIUM] Strange GitLab Reporting - Quota is allocated to GitLab Resources
Bug Reporting Resource_Types
coscine/issues#2412 2

[HIGH][LIVE] Suspicious Quota numbers in project
Bug Quota_Management
coscine/issues#2418 Tomorrow 3

Status_Doing 6 13

Allow users to sort search results
Review_Technical_ Search_&_Discoverability
coscine/issues#2264 3

[LIVE][MEDIUM]Search Categories/counts are broken
Bug Search_&_Discoverability
coscine/issues#2373 3

[LOW][LIVE] Elastic Search: User can not get a search result for specific names
Bug Search_&_Discoverability
coscine/issues#2317 3

Hint the user on how organization affiliation influences the visible resource types
Organizational_Structure UI_&_UX
coscine/issues#2393 1

Resources and Project API errors due to UDE being hacked
Bug Search_&_Discoverability

Status_Peer-Review_Testing 1 5

Include the new metadata structure in all interaction points
Metadata_Management
coscine/issues#1792 5

Closed 3 2

[HIGH][LIVE] Admin page quota problem
Bug Status_Deployed_to_D-SP10_ Status_Deployed_to_Live_ Status_Released_ UI_&_UX
coscine/issues#2427

[LIVE][HIGH] Upload/Update button always enabled with empty required fields for resource types as LinkedData, gitlab and RDS-S3
Bug RCV Resource_Types Status_Deployed_to_D-SP10_ Status_Deployed_to_Live_ Status_Released_ Status_Works on LIVE Status_Works_on_D-SP10
coscine/issues#2417 2

[HIGH][LIVE] RWTH-RDS-S3 gone
Bug Resource_Types Status_Deployed_to_D-SP10_ Status_Deployed_to_Live_ Status_Released_
coscine/issues#2428

Figure 1: A scrum board is used to visualize overall progress. Additional labels provide an overview of project area (light blue), criticality (red), and deployment status (turquoise).

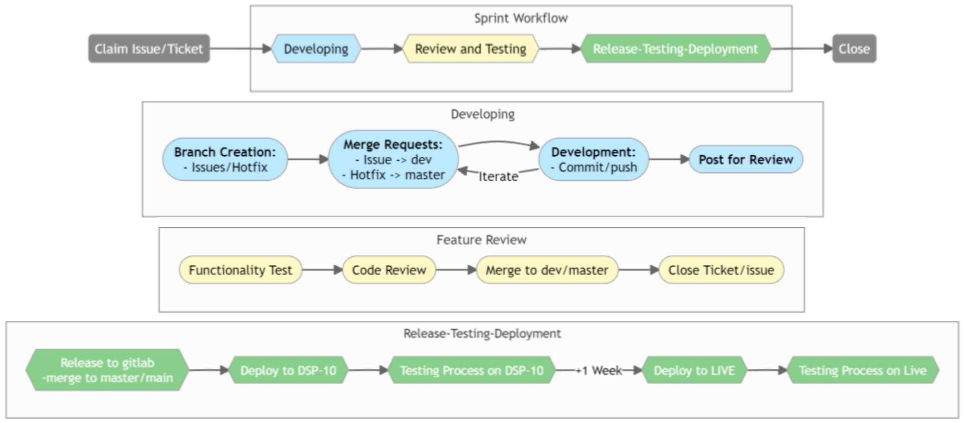


Figure 2: A three stage development process with feature branches using a develop (blue), a review (yellow) and a deployment (green) stage.

configuration files, monitor operational system parameters, or for assessing the quality of research data (Cimiano, Pietsch, & Wiljes, 2021). A non-code example is testing user-provided configuration files with basic syntactic checks written to generate a JUnit XML output which in turn can be visualized in detail by GitLab[3]. This pattern can be easily reproduced for comparable situations, where XML or JSON configuration files need to be edited by hand and can prevent accidental server downtimes due to configurations that cannot be parsed. Apart from correctness there are other quality checks like "Static Application Security Testing" or "Dependency Scanning" to protect against common security flaws.

This is strongly connected to operations and deployment tasks. It shows an evident interface within teams that work with a DevOps methodology. Nevertheless, a shared validation pipeline can significantly improve trust between operations and development teams, even in more traditionally organized teams with clear separation of concerns.

In addition to the source code, GitLab allows the storage of packages and containers that various package managers can retrieve, e.g., npm, NuGet, PyPI, Docker, or Podman. Private package registries form a natural point for handing over-developed applications or libraries to potential users or operations departments. While for most package managers, free public cloud services exist, many of these require a paid subscription if packages should not be publicly available. As a result, most of our teams have adopted building artifacts from their source code and uploading their results into GitLab. Using CI pipelines to perform these builds makes them widely reproducible thus combating a major challenge in scientific software development (Collberg & Proebsting, 2016; Hettrick, 2016). Making centrally and automatically built packages the distribution releases (of software, configuration files, or LaTeX documents) also eliminates effects from individually configured development environments.

2.3 On Interaction

While most teams use Issues and Epics already introduced above for their internal communication and planning, these features also provide strong measures for communication outside of the team. While this is not as widespread as internal planning, creating relatable descriptions can help stakeholders and users understand the challenges a certain feature poses on a development team. Additionally, making internal planning and discussion processes visible to stakeholders can lead to more comprehensible decision and prioritization processes.

Especially when working with various stakeholders, decision processes need to be consistent, fair, and understandable outside of the core team. To achieve this aim, one of our teams implemented and

[3] The pipeline depicted is part of one of our open-source projects and can be openly inspected here: https://git.rwth-aachen.de/coscine/graphs/applicationprofiles/-/pipelines/919925/

adapted the Scaled Agile Framework (SAFe)[4]. SAFe defines a transparent workflow for Epics from their initiation by a stakeholder or a team member using a template (see above), via an evaluation of a steering board and the technical analysis by a team member to the actual implementation. At each step of the Epic life-cycle (idea tank, funnel, review, analyzing, backlog, and in the process), the minimally necessary amount of information and work is invested and reviewed by the steering board before proceeding to the next step. Each review of the steering board is done using planning poker to achieve a consensus in the evaluation (Gandomani, Faraji, & Radnejad, 2019). The division into different Epic life-cycle steps prevents the development team from investing in lengthy implementation work before potential use cases and stakeholders are assessed. At the same time, the status of Epics is visible to stakeholders on the associated SAFe Kanban by using labels for each step. After the analysis, Epics move to the backlog and are prioritized for actual development. For Epic prioritization, the associated business value b, risk reduction r, time criticality c, effort or job size s and a strategic fit f are rated by the steering board meeting. Finally, the strategic fit allows slight adoption with respect to the long-term roadmap of the product and its importance to other strategic goals.

From these factors the weighted shortest job first metric m is computed:

$$m = \frac{b + r + c}{s} \cdot f \text{ with } b, r, c, s \in \{1, 2, 3, 5, 8, 13, 20\} \text{ and } f \in [0.8, 1.2]$$

The higher the value of m, the higher it is positioned on the SAFe Kanban board and the more likely it is to be considered next in software development. While the formula is quite simplistic, it uses elementary coherences: higher the business value, risk reduction, or time criticality, the more important it is to invest work. On the other hand, the more effort an Epic takes, the less attractive it becomes.

Epics within SAFe mainly consider long- and intermediate-term developments and communication with stakeholders on a strategic level. If stakeholders should also be involved in day-to-day business, Merge Requests (MR) can be used as another means of communication. MRs can be used to discuss smaller changes or additions to the source code. In one of our web platform projects, for example, users are offered the possibility to contribute their own configuration files, which are then served centrally for all platform users. To moderate the processes of adding or changing such a configuration, the team uses MRs that can be opened by the users themselves or through a feedback mechanism within the application. Within GitLab, MRs can then be used for discussions, social reactions, and show validations using previously configured CI jobs as discussed above.

2.4 On Operations

Additionally, in software development, a common task is the management of IT infrastructures (sometimes for development purposes itself, sometimes to provide a service for stakeholders. GitLab propagates the infrastructure as code (IaC) approach to make the provisioning of (virtual) infrastructures reproducible. While most of our teams' production environments are still hand-crafted and only partially automated, Terraform is introduced into the process for development environments. While Terraform is a stand-alone tool for IaC that can be managed easily from within Git, GitLab integrates with terraform such that the persistent state of the deployed infrastructure is saved within the project. New development environments can be deployed within a few seconds, given a compatible VM hosting service (in our case an Open Stack public cloud offer (Politze, 2022)). During infrastructure deployment, we use GitLab APIs to read information from the current user, project, or group to provision users with permissions directly to the VM environments using already provided SSH keys within GitLab[5], eventually eliminating the need for small development LDAPs. Additionally, the version tracking of Git repositories allows us to trace changes in the infrastructure. To enact the IaC, GitLab uses the same infrastructure that is also used to run CI tasks.

[4] see https://www.scaledagileframework.com/

[5] Using a barely documented key import URL like here: https://git.rwth-aachen.de/mpolitze.keys

On the other hand, not all operations related tasks are based on these cases. GitLab offers time-based schedules that trigger jobs. Again, these jobs use the same runners described previously for CI jobs. In our teams', scheduled jobs are mostly used where no explicit event triggers are available like polling the most recent versions of external sources like the ROR database[6], Overleaf projects or validating the connectivity to external services.

2.5 On Deployment

Once development (or other work) on an increment of the product has finished, the results need to be distributed. Again, this is an automation task in software engineering commonly called continuous delivery (CD). As the previously described automation tasks like CI or Schedules, GitLab uses a runner infrastructure to orchestrate and operate CD jobs. In contrast to the previously mentioned automation jobs, CD jobs usually relate to an environment. GitLab allows the creation of multiple environments from a single project. In the software-development-life-cycle, these are commonly used for integration, or staging or productive deployments. However, environments can also be used to reference multiple versions of the application deployed for different customers or smaller parts of the application. The latter is a pattern used by a team that uses environments to reference deployments of small application containers, the definitions of which are stored within the Git repository (cf. Figure 3).

Running services based on containers is further supported by a (private) container registry that can be easily referenced from compose files or helm charts. This primarily allows the creation of deployments on otherwise isolated servers. As with other public containers registries like DockerHub or Quay.io, it is important to retain certain hygiene when creating and naming containers. Again, automation can help solving some of these issues by creating tags for container images:

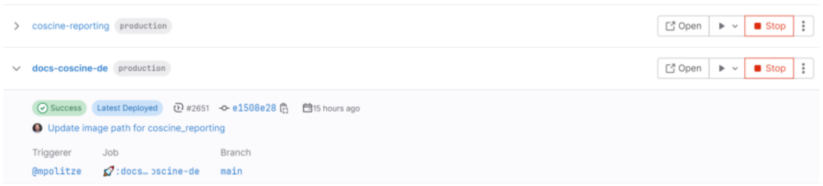
- for all commits based on the Git commit hash
- for Git tags (maybe only if matching version patterns like v3.0.10)
- a "latest" tag for the last commit on the main branch

This strategy gives very consistent tags and allows explicitly referencing releases by referencing Git tags. Workflows hence are more reproducible, especially when compared to referencing the volatile "latest" tag.

Running an application on a server that supports container environments like Docker or Kubernetes is very versatile, however for simple setups, this may require significant overhead to maintain the server and to build the containers. For simple (static) web pages GitLab's Pages feature is used often to display information. Combined with schedules this allows deploying regular updates to the page (Flemisch, Gläser, & Politze, 2020).

3 Remarks

The presented workflows are neither specific nor exclusive to GitLab. They can, at least partially, be reproduced by various other git-based software-life-cycle platforms like GitHub, BitBucket, Azure DevOps, or Gitea or even on generic project management platforms like OpenProject. However, we



#2651 e1508e28 15 hours ago

Update image path for coscine_reporting

Triggerer: @mpolitze
Job: :docs… ɔscine-de
Branch: main

Figure 3: Environments reference automation actions to deploy or stop instances.

[6] see https://ror.org

chose GitLab because it offers a good ratio between general project management and software-life-cycle management, and it has an extremely active community to learn from.

Some teams have gone beyond adopting the measures discussed earlier in the software-development-life-cycle and have shifted towards open-source software for their development toolchain and produced software. One of the first steps in this direction was the migration from proprietary version control systems to GitLab (Politze & Christoph, 2020). The assumption that all code will eventually be openly accessible has led to a heightened awareness of separating secrets from code and avoiding situations where security is falsely secured through obscurity. Making source code openly accessible has also enabled stakeholders like students or researchers to make contributions, which can enhance customer satisfaction[7]. Moreover, teams are encouraged to contribute to open-source projects used during development or for building services, but this may significantly impact their working processes.

GitLab is highly effective for internal collaboration, especially self-hosted instances, but each installation forms an isolated island open only to registered users. This poses a challenge when contributors from multiple organizations need to collaborate, and cross-organizational teams are often driven to the public cloud instance at Gitlab.com. Nonetheless, community cloud instances are essential to maintain self-sovereignty for source-code hosting.

Having a standard software development process helps teams when hiring new members, which is a common task for many organizations with limited term contracts from third-party funding. GitLab provides multiple functionalities to support distinct aspects of the development process, and teams can easily adopt them separately to address specific challenges such as stakeholder communication, hierarchical planning, and access management. It is also recommended to use GitLab's API, especially within its own automation frameworks, such as CI, Schedules, and CD. GitLab has become a crucial part of infrastructure compared to its previous use as just a hosting platform for Git repositories. Therefore, it is necessary to maintain a high standard of operation, including enforcing best practices such as two-factor authentication, access token expiry, and password policies to ensure availability and security. It is important to note that Git repositories are not suitable for all types of data, and large binaries should be handled with care by using appropriate extensions or artifact stores such as releases, packages, or container images. The use of Git also helped teams adopt open and text-based, data-driven formats for various tasks such as Markdown, Mermaid, Draw.io, and Gitbook.

4 Outlook

Most teams and projects only implement a fraction of the presented aspects, but they all establish a continuous improvement process to share experiences and enhance their working processes. While some aspects have been identified by at least one team, there are others yet to be discovered. Initially, all teams used legacy projects based on traditional, monolithic architectures, where single applications were installed on manually configured virtual machines. With the support of GitLab's software life-cycle adaptation, teams can gradually change their underlying technologies towards Infrastructure as Code (IaC) and container-based deployments and become more skilled in normalized workflows. This allows for an open contribution from stakeholders, directly involving them in the development process. For team success, central project management tools are crucial. However, choosing such a platform requires consideration of factors such as security, availability, and confidentiality. To address this, an optimal solution would be a community cloud service open for contributors from a broad academic community. Git.nrw aims to create such a service based on GitLab for collaboration, which is attached to eduGAIN and widely open for collaboration.

[7] Sometimes referred to as IKEA effect: https://en.wikipedia.org/wiki/IKEA_effect

5 Acknowledgements

The work was partially supported with resources granted by NFDI4Ing, funded by Deutsche Forschungsgemeinschaft (DFG) under project number 442146713, NFDI-MatWerk, funded by Deutsche Forschungsgemeinschaft (DFG) under project number 460247524, and FAIR Data Spaces, funded by the German Federal Ministry of Education and Research (BMBF) under funding reference FAIRDS11.

6 References

Cimiano, P., Pietsch, C., & Wiljes, C. (2021). *Studies in Analytical Reproducibility: the Conquaire Project.* Universität Bielefeld. doi:10.4119/unibi/2942780

Collberg, C., & Proebsting, T. A. (2016). Repeatability in computer systems research. *Communications of the ACM, 59*, 62–69. doi:10.1145/2812803

Colomb, J. (2020). GIN-tonic: Collaboration in research made easy. *GIN-tonic: Collaboration in research made easy*. Zenodo. doi:10.5281/zenodo.4159325

Cyra, M. A., Politze, M., & Timm, H. (2022). push for better RDM. *Bausteine Forschungsdatenmanagement*(2). doi:10.17192/BFDM.2022.2.8435

Flemisch, B., Gläser, D., & Politze, M. (2020). Metadaten und wo sie zu finden sind & Automatisierung, Qualitätssicherung, Zeitpläne: Praxiseinsatz für den GitLab Runner. *NFDI4Ing Virtual Community Meeting.*

Gandomani, T. J., Faraji, H., & Radnejad, M. (2019). Planning Poker in cost estimation in Agile methods: Averaging Vs. Consensus. *5th Conference on Knowledge Based Engineering and Innovation (KBEI).* IEEE. doi:10.1109/kbei.2019.8734960

Hahn, U., Hermann, S., Enderle, P., Fritze, F., Gärtner, M., & Kushnarenko, V. (2017). RePlay-DH - Realisierung einer Plattform und begleitender Dienste zum Forschungsdatenmanagement für die Fachcommunity - Digital Humanities. *RePlay-DH - Realisierung einer Plattform und begleitender Dienste zum Forschungsdatenmanagement für die Fachcommunity - Digital Humanities*. Heidelberg University Library. doi:10.11588/heidok.00022886

Halchenko, Y., Meyer, K., Poldrack, B., Solanky, D., Wagner, A., Gors, J., . . . Hanke, M. (2021). DataLad: distributed system for joint management of code, data, and their relationship. *Journal of Open Source Software, 6*, 3262. doi:10.21105/joss.03262

Hettrick, S. (2016). Research Software Sustainability. *Research Software Sustainability*. Edinburgh, United, Kingdom. Abgerufen am 30. December 2016

Küppers, B., Politze, M., & Schroeder, U. (2017). Reliable e-Assessment with GIT – Practical Considerations and Implementation. In R. Vogl (Hrsg.), *Proceedings of the 23rd EUNIS Congress*, (S. 253–262). Münster, Germany.

Politze, M. (2022). Hybrid Cloud Scaleout: Orchestrating Workloads with GitLab. In J.-F. Desnos, R. Yahyapour, & R. Vogl (Hrsg.), *EPiC Series in Computing: Proceedings of EUNIS 2022 – The 28th International Congress of European University Information Systems. 86.* EasyChair. doi:10.29007/nwh7

Politze, M., & Christoph, U. (2020). Migrating from Team Foundation Server to GitLab – A Progress Report. In *Proceedings of the EUNIS 2020 Congress* (S. 51–53). Helsinki, Finland.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., . . . Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data, 3*. doi:10.1038/sdata.2016.18

Yazdi, M., & Politze, M. (2020). Reverse Engineering the University Distributed Services. *Proceedings of the Future Technologies Conference (FTC)* (S. 223-238). Springer.

7 Authors' biographies

Dr. Marius Politze is head of the department "Research Process and Data Management" at the IT Center of RWTH Aachen University. Before that he held various posts at the IT Center as software developer, software architect and as a teacher for scripting and programming languages. His research focuses on Semantic Web, Linked Data, and architectures for distributed and service-oriented systems in the area of research data management. *(CRediT: Supervision, Software, Writing – original draft)*

Bernd Decker is deputy head of division of IT process support division at IT Center RWTH Aachen University since 2011. From 2006 to 2009 he worked at IT Center as Software Developer and since 2009 he is lead of the development group. His work is focused on IT solutions for processes in the field of E-Learning, E-Services, and campus management systems. *(CRediT: Conceptualization)*

Uta Christoph is deputy team lead of the group "Process and Application Development for Teaching" at the IT Center of RWTH Aachen University since 2018. She has worked at RWTH Aachen University as Project Manager and Software Developer since 2014. From 2012 to 2014 she was an international consultant for airport and cargo process optimization with Inform GmbH. Her work is now focused on the IT support of the quality management and accreditation process of RWTH Aachen University. *(CRediT: Conceptualization, Investigation)*

Petar Hristov is a full-stack developer in the department "Research Process and Data Management" at the IT Center of RWTH Aachen University. He has been working on the Data Management Platform Coscine since early 2021. Leveraging his rich background in mechanical and automotive engineering he creates a bridge between ongoing engineering-oriented projects and his daily work. He received his M.Sc. in Automotive Engineering at RWTH Aachen University in 2019. *(CRediT: Investigation, Software)*

Dr. Ilona Lang is the group lead of the group "Data Management Platform Coscine" which is part of the department "Research Process and Data Management" at the IT Center of RWTH Aachen University since 2022. From 2021 to 2022 she was the Service Manager of Coscine and from 2019 till 2021 a Research Data Manager in different RDM projects at the KIM in at the University of Konstanz. Her work is focused on the ongoing strategic and user-orientated development of Coscine while coordinating the demands of different stakeholders. *(CRediT: Conceptualization, Investigation, Writing – review & editing)*

Marcel Nellesen is the group lead of the group "Research Data Processes for HPC Systems" which is part of the department "Research Process and Data Management" at the IT Center of RWTH Aachen University since 2022. He started working at the IT Center as a software developer in 2013. His focus is on improving the connection between research data management and high-performance computing and providing a platform for the creation, review, and management of applications for storage and computation resources. *(CRediT: Conceptualization, Investigation)*

M. Amin Yazdi is a Ph.D. candidate in Data Science and Process Mining. His research focuses on introducing frameworks for modeling user interactions and enabling operational support to enhance User Experience, discover implicit user requirements, and facilitate the reusability of research data for Open Science Platforms via recommender systems. Since 2015, his professional expertise has been establishing and maintaining agile project management (Scrum) and leading the development team to go from conceptualization to realization of IT projects. He received his M.Sc. in Media Informatics at RWTH Aachen University with a major in Human-Computer Interaction (HCI). *(CRediT: Conceptualization, Investigation, Writing – review & editing)*