


Evaluation of tools for describing, reproducing and reusing scientific workflows


Philipp Diercks  ¹

Dennis Gläser  ²

Ontje Lünsdorf  ³

Michael Selzer  ⁴

Bernd Flemisch  ²

Jörg F. Unger  ¹


1. Department 7.7 Modeling and Simulation, Bundesanstalt für Materialforschung und -prüfung (BAM), Berlin.
2. Institut für Wasser- und Umweltsystemmodellierung, University of Stuttgart, Stuttgart.
3. Institut für Vernetzte Energiesysteme, Deutsches Zentrum für Luft- und Raumfahrt, Oldenburg.
4. Institut für Nanotechnologie, Karlsruher Institut für Technologie, Karlsruhe.



Date Received:

2022-12-05

Licenses:

This article is licensed under: 

Keywords:

FAIR, reproducibility, scientific workflows, tool comparison, workflow management

Data availability:

Data can be found here:

<https://doi.org/10.5281/zenodo.7790634>

Software availability:

Software can be found here:

<https://github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements>

Abstract. In the field of computational science and engineering, workflows often entail the application of various software, for instance, for simulation or pre- and postprocessing. Typically, these components have to be combined in arbitrarily complex workflows to address a specific research question. In order for peer researchers to understand, reproduce and (re)use the findings of a scientific publication, several challenges have to be addressed. For instance, the employed workflow has to be automated and information on all used software must be available for a reproduction of the results. Moreover, the results must be traceable and the workflow documented and readable to allow for external verification and greater trust. In this paper, existing workflow management systems (WfMSs) are discussed regarding their suitability for describing, reproducing and reusing scientific workflows. To this end, a set of general requirements for WfMSs were deduced from user stories that we deem relevant in the domain of computational science and engineering. On the basis of an exemplary workflow implementation, publicly hosted at GitHub (<https://github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements>), a selection of different WfMSs is compared with respect to these requirements, to support fellow scientists in identifying the WfMSs that best suit their requirements.

1 Introduction

With increasing volume, complexity and creation speed of scholarly data, humans rely more and more on computational support in processing this data. The “FAIR guiding principles for scientific data management and stewardship” [42] were introduced in order to improve the ability of machines to automatically find and use that data. FAIR comprises the four foundational principles “that all research objects should be *F*indable, *A*ccessible, *I*nteroperable and *R*eusable

7 (FAIR) both for machines and for people”. In giving abstract, high-level and domain-independent
8 guidelines, the authors answer the question of what constitutes good data management. However,
9 the implementation of these guidelines is still in its infancy, with many challenges not yet
10 identified and some of which may not have readily available solutions [31]. Furthermore, efforts
11 are made towards an Internet of FAIR Data and Services (IFDS) [17], which requires not only
12 the data, but also the tools and (compute) services to be FAIR.

13 Data processing is usually not a single task, but in general (and in particular for computational
14 simulations) relies on a chain of tools. Thus, to achieve transparency, adaptability and repro-
15 ducibility of (computational) research, the FAIR principles must be applied to all components
16 of the research process. This includes the tools (i. e. *any* research software) used to analyze the
17 data, but also the scientific workflow itself which describes how the various processes depend
18 on each other. In a community-driven effort, the FAIR principles were applied to research
19 software and extended to its specific characteristics by the FAIR for Research Software Working
20 Group [9]. For a discussion of how the FAIR principles should apply to workflows and workflow
21 management systems (WfMSs) we refer to [20].

22 In addition, in recent years there has been a tremendous development of different tools (see
23 e. g. <https://github.com/pditommaso/awesome-pipeline>) that aid the definition and
24 automation of computational workflows. These WfMSs have great potential in supporting
25 the goal above which is further discussed in section 1.1. The key features of WfMSs are also
26 highlighted in the context of bioinformatics workflows by [45], which compare several WfMSs
27 regarding aspects ranging from portability over scalability to the availability of learning resources.
28 A discussion of strengths and weaknesses of a selection of tools in the context of material sciences
29 is given in a recent work [34].

30 In this work, we would like to discuss how WfMSs can contribute to the transparency, adaptability,
31 reproducibility and reusability of computational research. Similar to [34, 45], we evaluate a
32 selection of WfMSs regarding a set of requirements, taking into account different possible
33 scenarios in which WfMSs are employed. In contrast to [34], we consider generic scenarios that
34 are not tied to a specific research domain (see section 2), from which we derive requirements
35 on WfMSs that we deem relevant in those contexts (see section 3). This leads to a set of
36 requirements that overlap with the ones presented in [45], but include more specific aspects of
37 workflow definitions and their development process. While [9, 20] discuss properties of *FAIR*
38 research software and workflows on a rather high level, this work focuses on how concrete
39 features of WfMSs may contribute to a more *FAIR* research software landscape. However, with
40 the considered requirements, we focus on the aspects *reusability* and *interoperability*, since
41 *findability* and *accessibility* lie outside the responsibilities of a WfMS.

42 Several WfMSs are evaluated with respect to the requirements by means of an exemplary
43 workflow, which is described in section 4, in addition to the available online documentation (see
44 below). The evaluation is presented in section 5, with the aim to support fellow scientists in
45 identifying the tools that best suit their requirements. The list of tools selected for comparison
46 is subjective and certainly not complete. However, a GitHub repository [16] providing an
47 implementation of the exemplary workflow for all tools and a short documentation with a link
48 to further information was created, with the aim to continuously add more tools in the future.

49 Furthermore, by demonstrating how the different tools could be used, we would like to encourage
 50 people to use WfMSs in their daily work.

51 1.1 Introduction to workflow management systems

52 In this paper, we use the term *process* to describe a computation, that is, the execution of a
 53 program to produce output data from input data. A process can be arbitrarily complex, but
 54 from the point of view of the workflow, it is a single, indivisible step. A *workflow* describes
 55 how individual processes relate to each other. Software-driven scientific workflows are often
 56 characterized by a complex interplay of various pieces of software executed in a particular order.
 57 The output of one process may serve as input to a subsequent process, which requires them to
 58 be executed sequentially with a proper mapping of outputs to inputs. Other computations are
 59 independent of each other and can be executed in parallel. Thus, one of the main tasks of WfMSs
 60 is the proper and efficient scheduling of the individual processes.

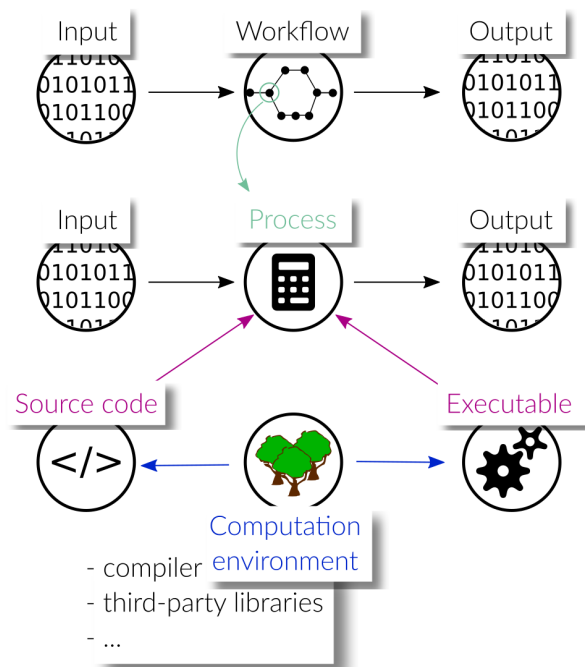


Figure 1: Schematic representation of software-driven scientific workflows. Note that a workflow as well as a process may have several inputs and outputs.

61 As shown in fig. 1, each process in the workflow, just as the workflow itself, takes some input to
 62 produce output data. A more detailed discussion of the different levels of abstractions related to
 63 workflows can be found in Griem et al. [21]. The behavior of a process is primarily determined
 64 by the source code that describes it, but may also be influenced by the interpreters/compiler
 65 used for translation or the machines used for execution. Moreover, the source code of a process
 66 may carry dependencies to other software packages such that the behavior of a process possibly
 67 depends on their versions. We use the term *computation environment* to collect all those software
 68 dependencies, that is, interpreters and/or compilers as well as third-party libraries and packages

69 that contribute to the computations carried out in a process. The exact version numbers of all
70 involved packages are crucial, as the workflow may not work with newer or older packages, or,
71 may produce different results.

72 As outlined in [30], WfMSs may be grouped into five classes. First, tools like *Galaxy* [1],
73 *KNIME* [8], and *Pegasus* [14] provide a graphical user interface (GUI) to define scientific
74 workflows. Thus, no programming skills are required and the WfMS is easily accessible for
75 everybody. With the second group, workflows are defined using a set of classes and functions
76 for generic programming languages (libraries and packages). This has the advantage that version
77 control (e. g. using *Git* (<https://git-scm.com>)) can be employed on the workflow. In addition,
78 the tool can be used without a graphical interface, e. g. in a server environment. Examples of
79 prominent tools are *AiiDA* [23, 39], *doit* [35], *Balsam* [33], *FireWorks* [24], *SciPipe* [28] and
80 *Guix Workflow Language* [46]. Third, tools like *Nextflow* [15], *Snakemake* [27], *Bpipe* [32] and
81 *Cluster Flow* [18] express the workflow using a domain specific language (DSL). A DSL is a
82 language tailored to a specific problem. In this context, it offers declarations and statements to
83 implement often occurring constructs in workflow definitions, which improves the readability
84 and reduces the amount of code. Moreover, the advantages of the second group also apply
85 for the third group. In contrast to the definition of the workflow in a programmatic way, the
86 fourth group comprises tools like *Popper* [25] and *Argo workflows* ([https://argoproj.g
87 ithub.io/argo-workflows/](https://argoproj.github.io/argo-workflows/)) which allow to specify the workflow in a purely declarative
88 way, by using configuration file formats like *YAML* [7]. In this case, the workflow specification
89 is concise and can be easily understood, but lacks expressiveness compared to the definition
90 of the workflow using programming languages. Fifth, there are system-independent workflow
91 specification languages like *CWL* [13] or *WDL* (<https://github.com/openwdl/wdl>). These
92 define a declarative language standard for describing workflows, which can then be executed by
93 a number of different engines like *Cromwell* [41], *Toil* [40], and *Tibanna* [29].

94 WfMSs can be used to create, execute and monitor workflows. They can help to achieve
95 reproducibility of research results by avoiding manual steps and automating the execution of
96 the individual processes in the correct order. More importantly, for a third person to reproduce
97 and reuse the workflow, it needs to be portable, that is, executable on any machine. Portability
98 can be supported by WfMSs with the integration of package management systems and container
99 technologies, which allow them to automatically re-instantiate the compute environment. Another
100 advantage of using WfMSs is the increase in transparency through a clear and readable workflow
101 specification. Moreover, after completion of the workflow, the tool can help to trace back
102 a computed value to its origin, by logging all inputs, outputs and possibly metadata of all
103 computations.

104 2 User stories

105 Starting from user stories that we consider representative for computational science and engi-
106 neering, a set of requirements is derived that serves as a basis for the comparison of different
107 WfMSs. In particular, a discussion on how the different tools implement these requirements is
108 provided.

109 Reproducibility, which is key to transparent research, is the main focus of the first user story

110 (see section 2.1). The second user story (see section 2.2) deals with research groups that develop
111 workflows in a joint effort where subgroups or individuals work on different components of the
112 workflow. Finally, the third user story focuses on computational research that involves generating
113 and processing large amounts of data, which poses special demands on how the workflow tools
114 organize the data that is created upon workflow execution (see section 2.3).

115 2.1 Transparent and reproducible research paper

116 *As a researcher, I want to share the code for my paper such that others are able to easily reproduce*
117 *my results.*

118 In this user story, the main objective is to guarantee the reproducibility of computational results
119 presented in scientific publications. Here, reproducibility means that a peer researcher is able to
120 rerun the workflow on some other machine while obtaining results that are in good agreement
121 with those reported in the publication. Mere reproduction could also be achieved without a
122 WfMS, e. g. by providing a script that executes the required commands in the right order, but this
123 comes with a number of issues that may be solved with a standardized workflow description.

124 First of all, reconstructing the logic behind the generation and processing of results directly from
125 script code is cumbersome and reduces the transparency of the research, especially for complex
126 workflows. Second, it is not straightforward for peer researchers to extract certain processes of a
127 workflow from a script and embed them into a different research project, hence the reusability
128 aspect is poorly addressed with this solution. Workflow descriptions may provide a remedy to
129 both of these issues, provided that each process in the workflow is defined as a unit with a clear
130 interface (see section 3.7).

131 While the workflow description helps peers to understand the details behind a research project,
132 it comes with an overhead on the side of the workflow creator, in particular when using a WfMS
133 for the first time. In the prevalent academic climate, but also in industrial (research) settings,
134 we therefore think that an important aspect of WfMSs is how easy they are to get started with
135 (see section 3.9). Similarly, if the WfMS provides a GUI to visualize and/or define the workflow
136 (see section 3.3), no special programming skills are required, which may be preferred due to the
137 easy access.

138 In the development phase, a workflow is typically run many times until its implementation is
139 satisfactory. With a scripted automation, the entire workflow is always executed, even if only one
140 process was changed since the last run. Since WfMSs have to know the dependencies between
141 processes, this opens up the possibility to identify and select only those parts of a workflow that
142 have to be rerun (see section 3.8). Besides this, the WfMS can display to the user which parts
143 are currently being executed, which ones have already been up-to-date, and which ones are still
144 to be picked (see section 3.2).

145 A general issue is that a workflow, or even each process in it, has a specific set of software- and
146 possibly hardware-requirements. This makes both reproducibility and reusability difficult to
147 achieve, especially over longer time scales, unless the computation environment in which the
148 original study was carried out is documented in a way that allows for a later re-instantiation.
149 The use of package managers that can export a given environment into a machine-readable

150 format from which they can then recreate that environment at a later time, may help to overcome
151 this issue. Another promising approach is to rely on container technologies. WfMSs have the
152 potential to automate the re-instantiation of a computation environment via integration of either
153 one of the above-mentioned technologies (see section 3.5). This makes it much easier for peers
154 to reproduce and/or reuse parts of a published workflow.

155 **2.2 Joint research (group)**

156 *As part of a research group, I want to be able to interconnect and reuse components of several*
157 *different workflows so that everyone may benefit from their colleagues' work.*

158 Similar to the previous user story, the output of such a workflow could be a scientific paper.
159 However, this user story explicitly considers interdisciplinary workflows in which the reusability
160 of individual components/modules is essential. Each process in the workflow may require a
161 different expertise and hence modularity and a common framework is necessary for an efficient
162 collaboration.

163 Many of the difficulties discussed in the previous user story are shared in a joint research project.
164 However, the collaborative effort in which the workflow description and those of its components
165 are developed promotes the importance of clear interfaces (see section 3.7) to ease communication
166 and an intuitive dependency handling mechanism (see section 3.5).

167 As mentioned in section 2.1, a GUI can help to increase the usability of a workflow for non-
168 programmers. However, in this user story it is important that the workflow definition is available
169 in a human-readable and manually editable format (see section 3.10). This facilitates version
170 control and the code review process as an essential aspect of teamwork.

171 Another challenge here is that such workflows often consist of heterogeneous models of dif-
172 ferent complexity, such as large computations requiring high-performance computing (HPC),
173 preprocessing of experimental data or postprocessing analyses. Due to this heterogeneity, it may
174 be beneficial to outsource computationally demanding tasks to HPC systems, while executing
175 cheaper tasks locally (see section 3.1). Workflows with such computationally expensive tasks
176 can also strongly benefit from effective caching mechanisms and the reuse of cached results
177 wherever possible (see section 3.8).

178 Finally, support for a hierarchical embedding of sub-workflows (possibly published and ver-
179 sioned) in another workflow is of great benefit as this allows for an easy integration of improve-
180 ments made in the sub-workflows by other developers (see section 3.6).

181 **2.3 Complex hierarchical computations**

182 *As a materials scientist, I want to be able to automate and manage complex workflows so I can*
183 *keep track of all associated data.*

184 Workflows in which screening or parameter sweeps are required typically involve running a large
185 number of simulations. Moreover, these workflows are often very complex with many levels of
186 dependencies between the individual tasks. Good data management that provides access to the
187 full provenance graph of all data can help to retain an overview over the large amounts of data

188 produced by such workflows (see section 3.4). For instance, the data management could be such
189 that desired information may be efficiently extracted via query mechanisms.

190 Another aspect regarding high-throughput computational screening is that the same computations
191 are carried out for many inputs (material structures) and the same workflow might be used for a
192 number of studies on varying input data. Here, a platform for publishing and sharing workflows
193 (see section 3.11) with the community can help to standardize and assure the quality of the
194 workflow. Furthermore, the findability and accessibility of workflows are increased, thereby
195 contributing to open science.

196 Due to the large amount of computationally demanding tasks in such workflows, it is helpful
197 if some computations can be outsourced to HPC systems (see section 3.1) with a clean way of
198 querying the current status during the typically long execution times (see section 3.2).

199 **3 Specific requirements on workflow management systems**

200 The user stories described above allow us to identify 11 requirements on WfMSs. Some of
201 these requirements concern the interaction with a WfMS from the perspective of a user of a
202 workflow, while others are related to the creation of a workflow definition and its readability or
203 portability. While portability is key to reproducible research, readability is an important aspect
204 of transparency. However, an easy and intuitive way of interacting with a WfMS is crucial for
205 workflows to be reused at all. Finally, reusability is enhanced if the workflow, or parts of it, can
206 be embedded into another workflow in a possibly different context. In the following, we will
207 describe the requirements in detail, as they will serve as evaluation criteria for the individual
208 WfMSs discussed in section 5.

209 **3.1 Support for job scheduling system**

210 As already mentioned, the main task of a WfMS is to automatically execute the processes of a
211 workflow in the correct order such that the dependencies between them are satisfied. However,
212 processes that do not depend on each other may be executed in parallel in order to speed up the
213 overall computation time. This requirement focuses on the ability of a WfMS to distribute the
214 computations on available resources. Job scheduling systems like e. g. Slurm (also commonly
215 referred to as batch scheduling or batch systems) are often used to manage computations to be
216 run and their resource requirements (number of nodes, CPUs, memory, runtime, etc.). Therefore,
217 it is of great benefit if WfMSs support the integration of widely-used batch systems such that
218 users can specify and also observe the used resources alongside other computations that were
219 submitted to their batch system in use. Besides this, this requirement captures the ability of a
220 WfMS to outsource computations to a remote machine, e. g. a HPC cluster or cloud. In this
221 sense, this requirement is crucial for workflows that require HPC resources to be reproducible.
222 For traditional HPC cluster systems it is usually necessary to transfer input and output data
223 between the local system and the cluster system. This can be done using the secure shell protocol
224 (SSH) and a WfMS may provide the automated transfer of a job's associated data. Ideally, the
225 workflow can be executed anywhere without changing the workflow definition itself, but only
226 the runtime arguments or a configuration file. The fulfillment of this requirement is evaluated by
227 the following criteria:

- 228 The workflow system supports the execution of the workflow on the local system.
- 229 The workflow system supports the execution of the workflow on the local system via
230 a batch system.
- 231 The workflow system supports the execution of the workflow via a batch system on
232 the local or a remote system.

233 3.2 Monitoring

234 Depending on the application, the execution of scientific workflows can be very time-consuming.
235 This can be caused by compute-intensive processes such as numerical simulations, or by a
236 large number of short processes that are executed many times. In both cases, it can be very
237 helpful to be able to query the state of the execution, that is, which processes have been finished,
238 which processes are currently being executed, and which are still pending. A trivial way of such
239 monitoring would be, for instance, when the workflow is started in a terminal which is kept
240 open to inspect the output written by the workflow system and the running processes. However,
241 ideally, the workflow system allows for submission of the workflow in the form of a process
242 running in the background, while still providing means to monitor the state of the execution. For
243 this requirement, two criteria are distinguished:

- 244 The only way to monitor the workflow is to watch the console output.
- 245 The workflow system provides a way to query the execution status at any time.

246 3.3 Graphical user interface

247 Independent of a particular execution of the workflow, the workflow system may provide
248 facilities to visualize the graph of the workflow, indicating the mutual dependencies of the
249 individual processes and the direction of the flow of data. One can think of this graph as the
250 template for the data provenance graph. This visualization can help in conveying the logic
251 behind a particular workflow, making it easier for other researchers to understand and possibly
252 incorporate it into their own research. The latter requires that the workflow system is able
253 to handle hierarchical workflows, that is, workflows that contain one or more sub-workflows
254 as processes (see section 3.6). Beyond a mere visualization, a GUI may allow for visually
255 connecting different workflows into a new one by means of drag & drop. We evaluate the
256 features of a graphical user interface by means of the following three criteria:

- 257 The workflow system provides no means to visualize the workflow
- 258 The workflow system or third-party tools allow to visualize the workflow definition
- 259 The workflow system or third-party tools provide a GUI that enables users to graphi-
260 cally create workflows

261 3.4 Data provenance

262 The data provenance graph contains, for a particular execution of the workflow, which data and
263 processes participated in the generation of a particular piece of data. Thus, this is closely related
264 to the workflow itself, which can be thought of as a template for how that data generation should

265 take place. However, a concrete realization of the workflow must contain information on the
 266 exact input data, parameters and intermediate results, possibly along with meta information on
 267 the person that executed the workflow, the involved software, the compute resources used and
 268 the time it took to finish. Collection of all relevant information, its storage in machine-readable
 269 formats and subsequent publication alongside the data can be very useful for future researchers
 270 in order to understand how exactly the data was produced, thereby increasing the transparency of
 271 the workflow and the produced data. Ideally, the workflow system has the means to automatically
 272 collect this information upon workflow execution, which we evaluate using the following criteria:

- 273 The workflow system provides no means to export relevant information from a partic-
 274 ular execution
- 275 The workflow system stores all results (also intermediate) together with provenance
 276 metadata about how they were produced

277 3.5 Compute environment




278 In order to guarantee interoperability and reproducibility of scientific workflows, the workflows
 279 need to be executable by others. Here, the re-instantiation of the compute environment (instal-
 280 lation of libraries or source code) poses the main challenge. Therefore, it is of great use if the
 281 WfMS is able to automatically deploy the software stack (on a per workflow or per process basis)
 282 by means of a package manager (e. g. conda <https://conda.io/>) or that running processes in
 283 a container (e. g. Docker <https://www.docker.com>, Apptainer <https://apptainer.org>
 284 (formerly Singularity)) is integrated in the tool. The automatic deployment of the software stack
 285 facilitates the execution of the workflow, and thus, greatly enhances its reproducibility. However,
 286 it does not (always) enable reuse, that is, the associated software can be understood, modified,
 287 built upon or incorporated into other software [9]. For instance, if a container image is used,
 288 it is important that the container build recipe (e. g. Dockerfile) is provided. This increases the
 289 reusability as it documents how a productive environment, suitable to execute the given workflow
 290 or process, can be set up. The author of the workflow, however, is deemed to be responsible for
 291 the documentation of the compute environment. For this requirement, we define the following
 292 evaluation criteria:

- 293 The automatic instantiation of the compute environment is not intended.
- 294 The workflow system allows the automatic instantiation of the compute environment
 295 on a per workflow basis.
- 296 The workflow system allows the automatic instantiation of the compute environment
 297 on a per process basis.

298 3.6 Hierarchical composition of workflows




299 A workflow consists of a mapping between a set of inputs (could be empty) and a set of outputs,
 300 whereas in between a number of processes are performed. Connecting the output of one workflow
 301 to the input of another workflow results in a new, longer workflow. This is particularly relevant
 302 in situations where multiple people share a common set of procedures (e. g. common pre- and
 303 postprocessing routines). In this case, copying the preprocessing workflow into another one is

304 certainly always possible, but does not allow to jointly perform modifications and work with
 305 different versions. Moreover, a composition might also require to define separate compute
 306 environments for each sub-workflow (e.g. using Docker/singularity or conda). Executing all
 307 sub-workflows in the same environment might not be possible because each sub-workflow might
 308 use different tools or even the same tools but with different versions (e. g. python2 vs. python3).
 309 Thus, WfMSs that can incorporate other workflows, possibly executed in a different compute
 310 environment, increase the reusability of a workflow substantially. This promotes the importance
 311 of supporting heterogeneous compute environments, which is reflected in the evaluation criteria
 312 for this requirement:

- 313  The workflow system does not allow the composition of workflows.
- 314  The workflow system allows to embed a workflow into another one for a single
 315 compute environment (homogeneous composition).
- 316  The workflow system allows to embed a workflow into another one for arbitrary many
 317 (on a per process basis) compute environments (hierarchical composition).

318 3.7 Interfaces

319 In a traditional file-based pipeline, the output files produced by one process are used as inputs to
 320 a subsequent process. However, it is often more convenient to pass non-file output (e. g. float or
 321 integer values) directly from one process to another without the creation of intermediate files. In
 322 this case, it is desirable that the WfMS is able to check the validity of the data (e. g. the correct
 323 data type) to be processed. Furthermore, this defines the interface for a process more clearly and
 324 makes it easier for someone else to understand how to use, adapt or extend the workflow/process.
 325 In contrast, in a file-based pipeline, this is usually not the case since a dependency in form of
 326 a file does not give information about the type of data contained in that file. For the sake of
 327 transparency and reusability, it is beneficial if a WfMS supports the definition of strongly-typed
 328 process interfaces. Type-checking the workflow definition before execution can also help to
 329 avoid unnecessary computations with erroneous workflows that attempt to transfer data with
 330 incompatible types. We distinguish these different types of interfaces by the following criteria:

- 331  The workflow system is purely file-based and does not define interface formats.
- 332  The workflow system allows for passing file and non-file arguments between processes.
- 333  The workflow system allows for defining strongly-typed process interfaces, supporting
 334 both file and non-file arguments.

335 3.8 Up-to-dateness

336 There are different areas for the application of workflows. On the one hand, people might use
 337 a workflow to define a single piece of reproducible code that, when executed, always returns
 338 the same result. Based on that, they might start a large quantity of different jobs and use the
 339 workflow system to perform this task. Another area of application is the constant development
 340 within the workflow (e.g. exchanging processes, varying parameters or even modifying the
 341 source code of a process) until a satisfactory result is obtained. The two scenarios require a

342 slightly different behavior of the workflow system. In the first scenario, all runs should be kept
 343 in the data provenance graph with a documentation of how each result instance has been obtained
 344 (e.g. by always documenting the codes, parameters, and processes). If identical runs (identical
 345 inputs and processes should result in the same output) are detected, a recomputation should be
 346 avoided and the original output should be linked in the data provenance graph. The benefit of
 347 this behavior certainly depends on the ratio between the computation time for a single process
 348 compared to the overhead to query the data base.

349 However, when changing the processes (e.g. coding a new time integration scheme or a new
 350 constitutive model), the workflow system should rather behave like a build system (such as make)
 351 - only recomputing the steps that are changed or that depend on these changes. In particular for
 352 complex problems, this allows to work with complex dependencies without manually triggering
 353 computations and results in automatically recomputing only the relevant parts. An example is a
 354 paper with multiple figures where each is a result of complex simulations that in itself depend on
 355 a set of general modules developed in the paper. The “erroneous” runs are usually not interesting
 356 and should be overwritten.

357 How this is handled varies between the tools, yielding the following evaluation criteria:


358 **R** The complete workflow is always **R**ecomputed.


359 **L** A new entry in the data provenance graph is created which **L**inks the previous result
 360 (without the need to recompute already existing results).


361 **U** Only the parts are recreated (**U**psdated) that are not up-to-date. This usually reduces the
 362 overhead to store multiple instances of the workflow, but at the same time also prevents -
 363 without additional effort (e.g. when executing in different folders) computing multiple
 364 instances of the same workflow.

365 3.9 Ease of first use

366 Although this is not a requirement per-se, it is beneficial if the workflow system has an intuitive
 367 syntax/interface and little work is required for a new user to define a first workflow. Research
 368 applications typically have a high intrinsic complexity, and therefore, the complexity added by the
 369 workflow management should be as small as possible. We note that this requirement is subjective
 370 and depends on the experience and skills of the user. Nevertheless, from the perspective of
 371 engineers and self-taught programmers, the following criteria are defined, considering aspects
 372 such as readability, expressiveness and knowledge of the tool:

373  difficult: Extensive knowledge of the tool and its design concepts as well as advanced
 374 programming skills are required to define a first workflow.

375  intermediate: Extensive knowledge of the tool and its design concepts and only basic
 376 programming skills are required to define a first workflow.

377  easy: Only basic programming skills are required to define a first workflow.

378 3.10 Manually editable workflow definition

379 While it can be beneficial to create and edit workflows using a GUI (see section 3.3), it may be
 380 important that the resulting workflow description is given in a human-readable format. This
 381 does not solely mean that the definition should be a text file, but also that the structure (e. g.
 382 indentation) and the naming are comprehensive. This facilitates version-controlling with git,
 383 and in particular the code review process. This increases the transparency of a workflow, and
 384 moreover, this does not force all users and/or developers to rely on the GUI. Evaluation criteria:

385 The workflow description is a binary file.

386 The workflow description is a text file but hard to interpret by humans.

387 The workflow description is a fully human-readable file format.

388 3.11 Platform for publishing and sharing workflows

389 The benefit of a workflow system is already significant when using it for individual research
 390 such as the development of an individual's paper or reproducing the paper that someone else has
 391 written, when their data processing pipeline is fully reproducible, documented and published.
 392 However, the benefit can be even more increased if people are able to jointly work on (sub-
 393)workflows together; particularly when a hierarchical workflow system is used. Even though
 394 workflows can easily be shared together with the work (e.g. in a repository), it might be beneficial
 395 to provide a platform that allows to publish documented workflows with a search and versioning
 396 functionality. This feature is not part of the requirement matrix to compare the different tools,
 397 but we consider a documentation of these platforms in the subsequent section as a good starting
 398 point for further research (exchange).

399 4 Exemplary workflow

400 A simple exemplary workflow was defined in order to analyze and evaluate the different WfMSs
 401 with respect to the requirements stated in section 3. This example is considered to be representa-
 402 tive for many problems simulating physical processes in engineering science using numerical
 403 discretization techniques. It consists of six steps, as shown in fig. 2:

404 1. generation of a computational mesh (Gmsh)

405 2. mesh format conversion (MeshIO)

406 3. numerical simulation (FEniCS)

407 4. post-processing of the simulation results (ParaView)

408 5. preparation of macro definitions (Python)

409 6. compilation of a paper into a *.pdf* file using the simulation results (Tectonic)

410 The workflow starts from a given geometry on which the simulation should be carried out and
 411 generates a computational mesh in the first step using Gmsh [19]. Here, the user can specify the
 412 size of the computational domain by a float value `domain_size`. The resulting mesh file format
 413 is not supported by FEniCS [4], which is the software that we are using for the simulation carried

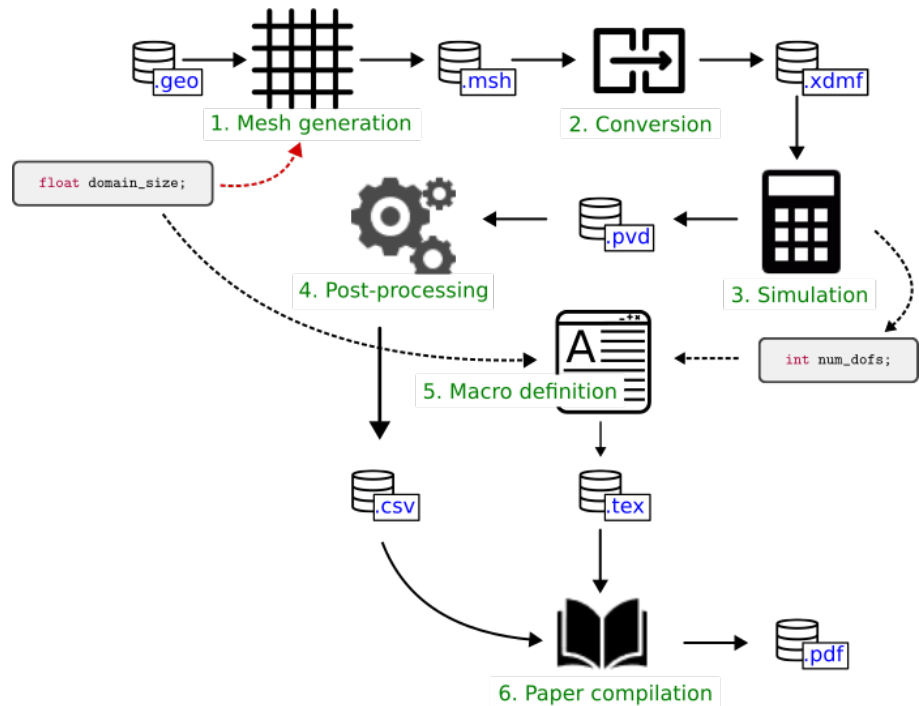


Figure 2: Task dependency graph of the exemplary workflow. Mapping of input and output data is indicated with black arrows with solid lines. A dashed line refers to non-file input or output (parameters). Here, red color is used to distinguish user input from automatic data transfer.

414 out in the third step. Therefore, we convert the mesh file in the second step of the workflow
 415 `.msh` to `.xdmf` using the python package MeshIO [36]. The simulation step yields result files in
 416 VTK file format [37] and returns the number of degrees of freedom used by the simulation as
 417 an integer value `num_dofs`. The VTK files are further processed using the python application
 418 programming interface (API) of ParaView [2], which yields the data of a plot-over-line of the
 419 numerical solution across the domain in `.csv` file format. This data, together with the values for
 420 the domain size and the number of degrees of freedom, is inserted into the paper and compiled
 421 into a `.pdf` file using the \LaTeX engine Tectonic [43] in the final step of the workflow.

422 Most steps transfer data among each other via files, but we intentionally built in the transfer of
 423 the number of degrees of freedom as an integer value to check how well such a situation can be
 424 handled by the tools. Example implementations of the exemplary workflow for various tools are
 425 available in a public repository [16].

426 5 Tool comparison

427 In this section, the selected WfMSs and their most important features are described and set in
 428 relation to the requirements defined in section 3. We note that to the best of our knowledge,
 429 existing add-on packages to the individual WfMSs are as well considered. As mentioned in the
 430 introduction, a large number of WfMSs exist, and the ones selected in this work represent only a
 431 small fraction of them. The considered WfMSs were selected on the basis of their popularity
 432 within the authors' communities, however, this has no implications on the quality of WfMSs

433 not considered in this work. As mentioned, we also plan to include implementations of the
434 exemplary workflow with further WfMSs in the online documentation in the future.

435 5.1 AiiDA

436 *AiiDA* [23, 39], the automated interactive infrastructure and database for computational science,
437 is an open source Python infrastructure. With *AiiDA*, workflows are written in the Python
438 programming language and managed and executed using the associated command line interface
439 “*verdi*”.

440 *AiiDA* was designed for use cases that are more focused on running heavy simulation codes
441 on heterogeneous compute hardware. Therefore, one of the key features of *AiiDA* is the HPC
442 interface. It supports the execution of (sub-) workflows on any machine and most resource
443 managers are integrated. In case of remote computing resources, any data transfer, retrieval and
444 storing of the results in the database or status checking is handled by the *AiiDA* daemon process.
445 Another key feature is *AiiDA*’s workflow writing system which provides strongly typed interfaces
446 and allows for easy composition and reuse of workflows. Moreover, *AiiDA* automatically keeps
447 track of all inputs, outputs and metadata of all calculations, which can be exported in the form of
448 provenance graphs.

449 *AiiDA*’s workflow system enables to easily compose workflows, but a general challenge seems
450 to be the management of the compute environment by the user. For external codes that do not
451 run natively in Python the implementation of so-called plugins is required. The plugin instructs
452 *AiiDA* how to run that code and might also contain (among other things) new data types or
453 parsers that are necessary, for instance, to validate the calculated results before storing them
454 in the database. Maintaining the plugin poses an additional overhead if the application code
455 changes frequently during development of the workflow. Moreover, the user has to take care of
456 the installation of the external code on the target computer.

457 However, since *AiiDA* version 2.1 it is possible to run code inside containers together with
458 any existing plugin for that code. This mitigates the issue of the manual installation of the
459 external code, but still requires a suitable plugin. Another benefit is that the information about the
460 compute environment is stored in the database as well. At the time of writing, the containerization
461 technologies Docker, Singularity <https://singularity-docs.readthedocs.io/en/latest/>
462 and Sarus <https://sarus.readthedocs.io/en/stable/> are supported.

463 In the special case of FEniCS (see section 4), which can be used to solve partial differential
464 equations and therefore covers a wide spectrum of applications, it is very difficult to define
465 a general plugin interface which covers all models. We note that due to this use case, which
466 is rather different from the use cases that *AiiDA* was designed for, the implementation of the
467 exemplary workflow (see section 4) uses “*aiida-shell*” [22], an extension to the *AiiDA* core
468 package which makes running shell commands easy. While this is convenient to get a workflow
469 running quickly, this leads to an undefined process interface since this was the purpose of the
470 aforementioned plugin for an external code. Considering the points above, compared to the other
471 tools, the learning curve with *AiiDA* is fairly steep.

472 In contrast to file-based workflow management systems, *AiiDA* defines data types for any data

473 that should be stored in a database. Consequently, non-file based inputs are well defined, but this
474 is not necessarily the case for file data. The reason for the choice of a database is that it allows
475 to query all stored data, and thus, enables powerful data analyses. For file-based workflows this
476 is difficult to reproduce, especially for large amounts of data.

477 In terms of the requirements defined in section 3, *AiiDA*'s strong points are execution, monitoring
478 and provenance. Due to the possibility to export provenance graphs, also level two of the
479 requirement *graphical user interface* is reached. Lastly, caching can be enabled in *AiiDA* to save
480 computation time. Caching in *AiiDA* means, that the database will be searched for a calculation
481 of the same hash and if this is the case, the same outputs are reused.

482 5.2 Common Workflow Language

483 “*Common Workflow Language (CWL)* [5] is an open standard for describing how to run command
484 line tools and connect them to create workflows” (<https://www.commonwl.org/>). One
485 benefit of it being a standard is that workflows expressed in *CWL* do not have to be executed by a
486 particular workflow engine, but can be run by any engine that is able to support the *CWL* standard.
487 In fact, there exist a number of workflow engines that support *CWL* workflows, e. g. the reference
488 implementation *cwltool* (<https://github.com/common-workflow-language/cwltool>),
489 *Toil* [40] or *StreamFlow* [10]. Note that so far we have tested our implementation only with
490 *cwltool*, however, in the evaluation we include all engines that support the *CWL* standard. That
491 is, in this work we consider that *CWL* fulfills a specific requirement if there exists an engine that
492 fulfills the requirement upon execution of a workflow written in *CWL*.

493 *CWL* was designed with a focus on data analysis using command line programs. To create a
494 workflow, each of the command line programs is “wrapped” in a *CWL* description, defining what
495 inputs are needed, what outputs are produced and how to call the underlying program. Typically,
496 this step also reduces the possibly large number of options of the underlying command line tool
497 to a few options or inputs that are relevant for the particular task of the workflow. In a workflow,
498 the wrapped command line tools can be defined as individual processes, and the outputs of
499 one process can be mapped to the inputs of other processes. This information is enough for
500 the interpreter to build up the dependency graph, and processes that do not depend on each
501 other may be executed in parallel. A process can also be another workflow, thus, hierarchical
502 workflow composition is possible. Moreover, there exist workflow engines (e. g. *Toil* [40] or
503 *StreamFlow* [10]) for *CWL* that support using job managers, for instance, Slurm [47].

504 The *CWL* standard also provides means to specify the software requirements of a process. For
505 instance, one can provide the URL of a Docker image or Docker file to be used for the execution
506 of a process. In case of the latter, the image is automatically built from the provided Docker file,
507 which itself contains the information on all required software dependencies. Besides this, the
508 *CWL* standard contains language features that allow listing software dependencies directly in
509 the description of a workflow or process, and workflow engines may automatically make these
510 software packages available upon execution. As one example, the current release of *cwltool*
511 supports the definition of software requirements in the form of e. g. *Conda* packages that are then
512 automatically installed when the workflow is run (see e. g. our implementation and the respective
513 pipelines at [16]).

514 In contrast to workflow engines that operate within a particular programming language, the
515 transfer of data from one process to another cannot occur directly via memory with *CWL*. For
516 instance, if the result of a process is an integer value, this value has to be read from a file produced
517 by the process, or, from its console output. However, this does not have to be done in a separate
518 process or by again wrapping the command line tool inside some script, since *CWL* supports the
519 definition of inline JavaScript code that is executed by the interpreter. This allows retrieving, for
520 instance, integer or floating point return values from a process with a small piece of code.

521 *CWL* requires the types of all inputs and outputs to be specified, which has the benefit that the
522 interpreter can do type checks before the execution of the workflow. A variety of primitive
523 types, as well as arrays, files or directories, are available. Files can refer to local as well as
524 online resources, and in the case of the latter, resources are automatically fetched and used upon
525 workflow execution.

526 There exist a variety of tools built around the *CWL* standard, such as the Rabix Composer (<https://rabix.io/>)
527 for visualizing and composing workflows in a GUI. Besides that and as mentioned
528 before, there are several workflow engines that support *CWL* and some of which provide extra
529 features. For instance, *cwltool* allows for tracking provenance information of individual workflow
530 runs. However, to the best of our knowledge, there exists no tool that automatically checks which
531 results are up-to-date and do not have to be reproduced (see section 3.8).

532 The *CWL* standard allows to specify the *format* of an input or output file by means of an *IRI*
533 (Internationalized Resource Identifier) that points to online-available resource where the file
534 format is defined. For processes whose output files are passed to the inputs of subsequent jobs,
535 the workflow engine can use this information to check if the formats match. To the best of our
536 knowledge, *cwltool* does so by verifying that the *IRIs* are identical, or performs further reasoning
537 in case the *IRIs* point to classes in ontologies (see, for instance, the class for the JSON file format
538 in the EDAM ontology at edamontology.org/format_3464). Such reasoning can make use of
539 defined relationships between classes of the ontology to determine file format compatibility and
540 thereby contribute to the requirement *process interfaces*. For more information on file format
541 specifications in *CWL* see commonwl.org/user_guide/topics/file-formats.html.

542 5.3 *doit*

543 “*doit* comes from the idea of bringing the power of build-tools to execute any kind of task” [35].
544 The automation tool *doit* is written in the Python programming language. In contrast to systems
545 which offer a GUI, knowledge of the programming language is required. However, it is not
546 required to learn an additional API since task metadata is returned as a Python dictionary.
547 Therefore, we consider this as very easy to get started quickly.

548 With *doit*, any shell command available on the system or python code can be executed. This
549 also includes the execution of processes on a remote machine, although all necessary steps (e. g.
550 connecting to the remote via SSH) need to be defined by the user. In general, such behavior
551 as described in section 3.1 is possible, but it is not a built-in feature of *doit*. Also, *doit* does
552 not intend to provide the compute environment. Therefore, while in general the composition of
553 workflows (see section 3.6) is easily possible via python imports, this only works for a single
554 environment. The status of the execution can be monitored via the console. Here, *doit* will skip

555 the execution of processes which are up-to-date and would produce the same result of a previous
556 execution. To determine the correct order in which processes should be executed, *doit* also
557 creates a directed acyclic graph (DAG) which could be used to visualize dependencies between
558 processes using “*doit-graph*” (<https://github.com/pydoit/doit-graph>), an extension to
559 *doit*. For each run (specific instance of the workflow), *doit* will save the results of each process
560 in a database. However, the tool does not provide control over what is stored in the database.
561 On the one hand, *doit* allows to pass results of one process as input to another process directly,
562 without creating intermediate files, so it is not purely file-based. On the other hand, the interface
563 for non-file based inputs does not define the data type.

564 5.4 Guix Workflow Language

565 The *Guix Workflow Language (GWL)* [46] is an extension to the open source package manager
566 GNU Guix [12]. *GWL* leverages several features from Guix, chief among these is the compute
567 environment management. Like Guix, *GWL* only supports GNU/Linux systems.

568 *GWL* can automatically construct an execution graph from the workflow process input/output
569 dependencies but also allows a manual specification. Support for HPC schedulers via DRMAA¹
570 is also available.

571 *GWL* doesn’t provide a graphical user interface, interactions are carried out using a command-line
572 interface in a text terminal. Monitoring is also only available in the form of simple terminal
573 output.

574 There is support to generate a GraphViz (see e. g. <https://graphviz.org>) description of the
575 workflow, which allows basic visualization of a workflow. Although not conveniently exposed²,
576 *GWL* has a noteworthy unique feature inherited from Guix: precise software provenance tracking.
577 Guix contains complete build instructions for every package (including their history through git),
578 which enables accounting of source code and the build process, like for example compile options,
579 of all tools used in the workflow. Integrity of this information is ensured through cryptographic
580 hash functions. This information can be used to construct data provenance graphs with a high
581 level of integrity (basically all userspace code of the compute environment can be accounted
582 for [11]).

583 *GWL* uses Guix to setup compute environments for workflow processes. Each process is
584 executed in an isolated³ compute environment in which only specified software packages are
585 available. This approach minimizes (accidental) side-effects from system software packages
586 and improves workflow reproducibility. Interoperability also benefits from this approach, since
587 a Guix installation is the only requirement to execute a workflow on another machine. As Guix
588 provides build instructions for all software packages, it should be easily possible to recreate
589 compute environments in the future, even if the originally compiled binaries have been deprecated
590 in the meanwhile (see [3] for a discussion about long-term reproducibility).

1. Distributed Resource Management Application API <https://www.drmaa.org>

2. *GWL* doesn’t provide a command to export provenance graphs in any way, instead Guix needs to be queried for build instruction, dependency graphs and similar provenance information of a workflows software packages

3. By default, lightweight isolation is setup by limiting the `PATH` environment variable to the compute environment. Stronger isolation via Linux containers is also optionally available.

591 Composition of workflows is possible, workflows can be imported into other workflows. Com-
592 position happens either by extracting individual processes (repurposing them in a new workflow)
593 or by appending new processes onto the existing workflow processes.

594 *GWL* relies exclusively on files as interface to workflow processes. There's no support to
595 exchange data on other channels, as workflow processes are executed in isolated environments.

596 Like other WfMSs, *GWL* caches the result of a workflow process using the hash of its input data.
597 If a cached result for the input hash value exists, the workflow processes execution is skipped.

598 *GWL* is written in the Scheme [38] implementation GNU Guile [44], but in addition to Scheme,
599 workflows can also be defined in *wisp* [6], a variant of Scheme with significant whitespace⁴.
600 *wisp* syntax thus resembles Python, which is expected to flatten the learning curve a bit for
601 scientific audience. However, error messages are very hard to read without any background in
602 Scheme. On first use, *GWL* will be very difficult in general. This problem is acknowledged by
603 the *GWL* authors and might be subject to improvements in the future.

604 As both *wisp* and Scheme code is almost free of syntactic noise in general, workflows are almost
605 self-describing and easily human-readable.

606 In summary, *GWL* provides a very interesting and sound set of features especially for repro-
607 ducibility and interoperability. These features come at the cost of a Guix installation, which
608 requires administrator privileges. The workflow language is concise and expressive, but error
609 messages are hard to read. At the current stage, *GWL* can only be recommended to experienced
610 scheme programmers or to specialists with high requirements on software reproducibility and
611 integrity.

612 5.5 Nextflow and Snakemake

613 With *Nextflow* [15] and *Snakemake* [30], the workflow is defined using a DSL which is an
614 extension to a generic programming language (Groovy for *Nextflow* and Python for *Snakemake*).
615 Moreover, *Nextflow* and *Snakemake* also allow to use the underlying programming language
616 to generate metadata programmatically. Thus, authoring scientific workflows with *Nextflow* or
617 *Snakemake* is very easy.

618 The process to be executed is usually a shell command or an external script. The integration
619 with various scripting languages is an import feature of *Snakemake* as well as *Nextflow*, which
620 encourages readable modular code for downstream plotting and summary tasks. Also boilerplate
621 code for command line interfaces (CLIs) in external scripts can be avoided. Another feature of
622 *Snakemake* is the integration of Jupyter notebooks, which can be used to interactively develop
623 components of the workflow.

624 Both tools implement a CLI to manage and run workflows. By default, the status of the execution
625 is monitored via the console. With *Nextflow*, it is possible to monitor the status of the execution
626 via a weblog. *Snakemake* supports an external server to monitor the progress of submitted
627 workflows.

4. *GWL* is not a workflow language in the strict sense. At its core, it is a Scheme library that defines functions and objects for workflow composition (like processes, inputs, outputs, etc.). It allows workflows to be defined in both Scheme and *wisp*.

628 With regard to the execution of the workflow (section 3.1), the user can easily run the workflow
629 on the local machine and the submission via a resource manager (e. g. Slurm, Torque) is integrated.
630 Therefore, individual process resources can be easily defined with these tools if the workflow is
631 submitted on a system where a resource manager is installed, i. e. on a traditional HPC cluster
632 system. Despite this, only level two of the defined criteria is met for *Nextflow*, since the execution
633 of the workflow on a remote machine and the accompanied transfer of data is not handled by the
634 tool. For *Snakemake*, if the CLI option “default-remote-provider” is used, all input and output
635 files are automatically down- and uploaded to the defined remote storage, such that no workflow
636 modification is necessary.

637 The requirement *up-to-dateness* is handled differently by *Nextflow* and *Snakemake*. By default,
638 *Nextflow* recomputes the complete workflow, but with a single command-line option existing
639 results are retrieved from the cache and linked such that a re-execution is not required. In
640 this case, *Nextflow* allows storing multiple instances of the same workflow upon variation of a
641 configuration parameter. *Snakemake* will behave like a build tool in this context and skip the
642 re-execution of processes whose targets already exist and update any process whose dependencies
643 have changed.

644 A strong point of *Nextflow* and *Snakemake* is the integration of the conda package management
645 system and container technologies like Docker. For example, the compute environment can be
646 defined for each process based on a conda environment specification file or a certain Docker
647 image. Upon execution of the workflow, the specified compute environment is re-instantiated
648 automatically by the WfMS, making it very easy to reproduce results of or built upon existing
649 workflows. Furthermore, since the tool is able to deploy the software stack on a per process
650 basis, the composition of hierarchical workflows as outlined in section 3.6 is possible.

651 Similar to *doit*, both tools do not provide a GUI to graphically create and modify workflows.
652 However, a visualization of the workflow, i. e. a dependency graph of the processes, can be
653 exported. Moreover, it is possible to export extensive reports detailing the provenance of the
654 generated data.

655 *Nextflow* and *Snakemake* can also be regarded as file-based workflow management systems.
656 Therefore, interface formats, i. e. class structures or types of the parameters passed from one
657 process to the subsequent one, are not clearly defined.

658 5.6 Evaluation matrix

659 The evaluation of the WfMSs provided in section 5 in terms of the requirements described
660 in section 3 on the example of the workflow outlined in section 4 yields the evaluation matrix
661 depicted in table 1.

662 6 Summary

663 In this work, six different WfMSs (*AiiDA*, *CWL*, *doit*, *GWL*, *Nextflow* and *Snakemake*) are studied.
664 Their performance is evaluated based on a set of requirements derived from three typical user
665 stories in the field of computational science and engineering. On the one hand, the user stories
666 are focusing on facilitating the development process, and on the other hand on the possibility of

Table 1: Evaluation of the considered workflow management systems.

Requirement	Workflow Management System					
	<i>AiiDA</i>	<i>CWL</i>	<i>doit</i>	<i>GWL</i>	<i>Nextflow</i>	<i>Snake- make</i>
Job scheduling system	●●●●	●●●○	●○○○	●●●○	●●●○	●●●●
Monitoring	●●	●●	●○	●○	●○	●○
Graphical user interface	●●○	●●●	●●○	●○○	●●○	●●○
Provenance	●●	●●	●○	●○	●●	●●
Compute environment	●●●●	●●●●	●○○○	●●●●	●●●●	●●●●
Composition	●●○	●●●	●●○	●●●	●●●	●●●
Process interfaces	●●○	●●●	●○○	●○○	●○○	●○○
Up-to-dateness	L	R	U	U	L	U
Ease of first use	●○○	●●○	●●●	●○○	●●●	●●●
Manually editable	●●●	●●●	●●●	●●●	●●●	●●●

667 reusing and reproducing results obtained using research software. The choice for one WfMS or
 668 the other is strongly subjective and depends on the particular application and the preferences of
 669 its developers. The overview given in table 1 together with the assessments in section 5 may
 670 only serve as a basis for an individual decision making.

671 For researchers that want to start using a WfMS, an important factor is how easy it is to get a first
 672 workflow running. We note again that the evaluation criteria of the requirement for the *ease of*
 673 *first use* are difficult to measure objectively and refer to section 3.9 for what is considered *easy*
 674 *to use* in this work. For projects that are written in Python, a natural choice may be *doit*, which
 675 operates in Python and is easy to use for anyone familiar with the language. Another benefit
 676 of this system is that one can use Python functions as processes, making it possible to easily
 677 transfer data from one process to the other via memory without the need to write and read to
 678 disk. In order to make a workflow portable, developers have to provide additional resources that
 679 allow users to prepare their environment such that all software dependencies are met, prior to
 680 the workflow execution.

681 To create portable workflows more easily, convenient tools are *Nextflow* or *Snakemake*, where
 682 one can specify the compute environment in terms of a conda environment file or a container
 683 image on a per-process basis. They require to learn a new domain-specific language, however,
 684 our assessment is that it is easy to get started as only little syntax has to be learned in order to get
 685 a first workflow running.

686 The strengths of *AiiDA* are the native support for distributing the workload on different (registered)
 687 machines, the comprehensive provenance tracking, and also the possibility to transfer data among
 688 processes without the creation of intermediate files.

689 *CWL* has the benefit of being a language standard rather than a specific tool maintained by a
 690 dedicated group of developers. This has led to a variety of tooling developed by the community
 691 as e. g. editors for visualizing and modifying workflows with a GUI. Moreover, the workflow
 692 description states the version of the standard in which it is written, such that any interpreter
 693 supporting this standard should execute it properly, which reduces the problem of version pinning

694 on the level of the workflow interpreter.

695 Especially for larger workflows composed of processes that are still under development, and
696 are thus changing over time, it may be useful to rely on tools that allow to define the process
697 interfaces by means of strongly-typed arguments. This can help to detect errors early on, e. g. by
698 static type checkers. *CWL* and *AiiDA* support the definition of strongly-typed process interfaces.
699 The rich set of options and features of these tools make them more difficult to learn, but at the
700 same time expose a large number of possibilities.

701 **7 Outlook**

702 This overview is not meant to be static, but we plan to continue the documentation online
703 in the git repository [16] that contains the implementation of the exemplary workflow. This
704 allows us to take into consideration other WfMSs in the future, and to extend the documentation
705 accordingly. In particular, we would like to make the repository a community effort allowing
706 others to contribute either by modifications of the existing tools or adding new WfMSs. All of
707 our workflow implementations are continuously and automatically tested using GitHub Actions
708 [https://github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements/ac](https://github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements/actions)
709 [tions](https://github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements/actions), which may act as an additional source of documentation on how to launch the workflows.

710 One of the challenges that we have identified is the use of container technology in the HPC
711 environment. In most cases, the way users should interact with such a system is through a module
712 system provided by the system administrators. The module system allows to control the software
713 environment (versioning, compilers) in a precise manner, but the user is limited to the provided
714 software stack. For specific applications, self-written code can be compiled using the available
715 development environment and subsequently run on the system, which is currently the state of
716 the art in using HPC systems. However, this breaks the portability of the workflow.

717 Container technology, employing the “build once and run anywhere” concept, seems to be a
718 promising solution to this problem. Ideally, one would like to be able to run the container
719 application on the HPC system, just as any other MPI-distributed application. Unfortunately,
720 there are a number of problems entailed with this approach.

721 When building the container, great care must be taken with regard to the MPI configuration,
722 such that it can be run successfully across several nodes. Another issue is the configuration of
723 Infiniband drivers. The container has to be build according to the specifics of the HPC system
724 that is targeted for execution. From the perspective of the user, this entails a large difficulty,
725 and we think that further work needs to be done to find solutions which enable non-experts in
726 container technology to execute containerized applications successfully in an HPC environment.

727 Furthermore, challenges related to the joint development of workflows became apparent. In this
728 regard, strongly-typed interfaces are required in order to minimize errors and transparently and
729 clearly communicate the metadata (inputs, outputs) associated with a process in the workflow.
730 This is recommended both for single parameters, but it would be also great to extend that idea
731 to files - not only defining the file type which is already possible within *CWL* - but potentially
732 allowing a type checking of the complete data structure within the file. However, based on our
733 experience with the selected tools, these interfaces and their benefits come at the cost of some

734 form of plugin or wrapper around the software that is to be executed, thus possibly limiting the
735 functionality of the wrapped tool. This means there is a trade-off between easy authoring of the
736 workflow definition (e. g. easily executing any shell command) and implementation overhead
737 for the sake of well-defined interfaces.

738 Another aspect is how the workflow logic can be communicated efficiently. Although all tools
739 allow to generate a graph of the workflow, the dependencies between processes can only be
740 visualized for an executable implementation of the workflow, which most likely does not exist
741 in early stages of the project where it is needed the most.

742 An important aspect is the documentation of the workflow results and how they have been
743 obtained. Most tools offer an option to export the data provenance graph, however it would be
744 great to define a general standard supported by all tools as e. g. provided by *CWLProv* [26].

745 A further direction of future research may also be a better measure for the ease of (first) use. As
746 stated in section 3.9 this is rather subjective and depends on the experience and skills of the user.
747 One could possibly treat this requirement statistically by carrying out a survey of the users of the
748 respective tools.

749 **Financial disclosure**

750 None reported.

751 **Conflict of interest**

752 The authors declare no potential conflict of interests.

753 **8 Acknowledgements**

754 The authors would like to thank the Federal Government and the Heads of Government of the
755 Länder, as well as the Joint Science Conference (GWK), for their funding and support within the
756 framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) -
757 project number 442146713. Moreover, we would like to thank Sebastian P. Huber, Michael
758 R. Crusoe, Eduardo Schettino, Ricardo Wurmus, Paolo Di Tommaso and Johannes Köster for
759 their valuable remarks and comments on an earlier version of this article and the workflow
760 implementations.

761 **9 Roles and contributions**

762 **Philipp Diercks:** Investigation; methodology; software; writing - original draft; writing - review
763 and editing.

764 **Dennis Gläser:** Investigation; methodology; software; writing - original draft; writing - review
765 and editing.

766 **Ontje Lünsdorf:** Investigation (supporting); software; writing - original draft (supporting).

767 **Michael Selzer:** Writing - review and editing (supporting).

768 **Bernd Flemisch:** Conceptualization (supporting); Funding acquisition; Project administration;
769 Writing - review and editing.

770 **Jörg F. Unger:** Conceptualization (lead); Funding acquisition; Project administration; Writing -
771 original draft (supporting); Writing - review and editing.

772 References

- 773 [1] Enis Afgan et al. “The Galaxy platform for accessible, reproducible and collaborative
774 biomedical analyses: 2018 update”. In: *Nucleic Acids Research* 46.W1 (May 2018),
775 W537–W544. ISSN: 0305-1048. DOI: [10.1093/nar/gky379](https://doi.org/10.1093/nar/gky379). eprint: <https://academic.oup.com/nar/article-pdf/46/W1/W537/25110642/gky379.pdf>. URL:
776 <https://doi.org/10.1093/nar/gky379>.
777
- 778 [2] James Ahrens, Berk Geveci, and Charles Law. “ParaView: An End-User Tool for Large-
779 Data Visualization”. In: *The Visualization Handbook*. Elsevier, 2005.
- 780 [3] Mohammad Akhlaghi et al. “Toward Long-Term and Archivable Reproducibility”. In:
781 *Computing in Science & Engineering* 23.3 (May 2021), pp. 82–91. ISSN: 1521-9615,
782 1558-366X. DOI: [10.1109/mcse.2021.3072860](https://doi.org/10.1109/mcse.2021.3072860). URL: <https://doi.org/10.1109/mcse.2021.3072860>.
783
- 784 [4] M.S. Alnaes et al. “The FEniCS Project Version 1.5”. In: *Archive of Numerical Software* 3
785 (2015). DOI: [10.11588/ans.2015.100.20553](https://doi.org/10.11588/ans.2015.100.20553).
- 786 [5] Peter Amstutz et al. *Common Workflow Language, v1.0*. <https://doi.org/10.6084/m9.figshare.3115156.v2>. July 2016. DOI: [10.6084/m9.figshare.3115156.v2](https://doi.org/10.6084/m9.figshare.3115156.v2).
787
- 788 [6] Arne Babenhauerheide. *SRFI 119: wisp: simpler indentation-sensitive scheme*. <https://srfi.schemers.org/srfi-119/>. June 2015.
789
- 790 [7] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. *YAML Ain't Markup Language (YAML)*
791 *version 1.2*. Accessed: 2022-08-31. Version 1.2. <https://yaml.org/spec/1.2.2/>.
792 2021.
- 793 [8] Michael R. Berthold et al. “KNIME: The Konstanz Information Miner”. In: *Data Analysis*
794 *, Machine Learning and Applications : Proceedings of the 31st Annual Conference of the*
795 *Gesellschaft für Klassifikation e. V., Albert-Ludwigs-Universität Freiburg, March 7-9 ,*
796 *2007*. New York: Springer, 2007.
- 797 [9] Neil P. Chue Hong et al. *FAIR Principles for Research Software (FAIR4RS Principles)*.
798 <https://doi.org/10.15497/RDA00068>. Version 1.0. May 2022. DOI: [10.15497](https://doi.org/10.15497/RDA00068)
799 [/RDA00068](https://doi.org/10.15497/RDA00068). URL: <https://doi.org/10.15497/RDA00068>.
- 800 [10] Iacopo Colonnelli et al. “StreamFlow: cross-breeding cloud with HPC”. In: *IEEE Trans-*
801 *actions on Emerging Topics in Computing* 9.4 (2021), pp. 1723–1737. DOI: [10.1109](https://doi.org/10.1109/TETC.2020.3019202)
802 [/TETC.2020.3019202](https://doi.org/10.1109/TETC.2020.3019202).
- 803 [11] Ludovic Courtès. “Building a Secure Software Supply Chain with GNU Guix”. In: *The*
804 *Art, Science, and Engineering of Programming* 7.1 (June 2022). ISSN: 2473-7321. DOI:
805 [10.22152/programming-journal.org/2023/7/1](https://doi.org/10.22152/programming-journal.org/2023/7/1). URL: [https://doi.org/10.2](https://doi.org/10.22152/programming-journal.org/2023/7/1)
806 [2152/programming-journal.org/2023/7/1](https://doi.org/10.22152/programming-journal.org/2023/7/1).

- 807 [12] Ludovic Courtès. “Functional Package Management with Guix”. In: *European Lisp*
808 *Symposium* (June 2013). DOI: [10.48550/ARXIV.1305.4584](https://doi.org/10.48550/ARXIV.1305.4584). URL: [https://arxiv](https://arxiv.org/abs/1305.4584)
809 [.org/abs/1305.4584](https://arxiv.org/abs/1305.4584).
- 810 [13] Michael R. Crusoe et al. “Methods included. standardizing computational reuse and
811 portability with the Common Workflow Language”. In: *Commun. ACM* 65.6 (June 2022),
812 pp. 54–63. ISSN: 0001-0782, 1557-7317. DOI: [10.1145/3486897](https://doi.org/10.1145/3486897). URL: [https://do](https://doi.org/10.1145/3486897)
813 [i.org/10.1145/3486897](https://doi.org/10.1145/3486897).
- 814 [14] Ewa Deelman et al. “Pegasus, a workflow management system for science automation”.
815 In: *Future Generation Computer Systems* 46 (2015), pp. 17–35. ISSN: 0167-739X. DOI:
816 [10.1016/j.future.2014.10.008](https://doi.org/10.1016/j.future.2014.10.008). URL: [https://www.sciencedirect.com/sci](https://www.sciencedirect.com/science/article/pii/S0167739X14002015)
817 [ence/article/pii/S0167739X14002015](https://www.sciencedirect.com/science/article/pii/S0167739X14002015).
- 818 [15] Paolo Di Tommaso et al. “Nextflow enables reproducible computational workflows”.
819 In: *Nat Biotechnol* 35.4 (Apr. 2017), pp. 316–319. ISSN: 1087-0156, 1546-1696. DOI:
820 [10.1038/nbt.3820](https://doi.org/10.1038/nbt.3820). URL: <https://doi.org/10.1038/nbt.3820>.
- 821 [16] Philipp Diercks et al. *NFDI4Ing Scientific Workflow Requirements*. Version 0.0.1. [https:](https://github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements)
822 [//github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements](https://github.com/BAMresearch/NFDI4IngScientificWorkflowRequirements). July
823 2022.
- 824 [17] European Commission and Directorate-General for Research and Innovation. *Realising*
825 *the European open science cloud : first report and recommendations of the Commission*
826 *high level expert group on the European open science cloud*. Publications Office, 2016.
827 DOI: [10.2777/940154](https://doi.org/10.2777/940154).
- 828 [18] Philip Ewels et al. “Cluster Flow: A user-friendly bioinformatics workflow tool [version 2;
829 referees: 3 approved].” In: *F1000Research* 5 (2016), p. 2824. DOI: [10.12688/f1000res](https://doi.org/10.12688/f1000research.10335.2)
830 [earch.10335.2](https://doi.org/10.12688/f1000research.10335.2). URL: <http://dx.doi.org/10.12688/f1000research.10335.2>.
- 831 [19] Christophe Geuzaine and Jean-François Remacle. “Gmsh: A 3-D finite element mesh
832 generator with built-in pre- and post-processing facilities. THE GMSH PAPER”. In:
833 *Int. J. Numer. Meth. Engng.* 79.11 (May 2009), pp. 1309–1331. ISSN: 0029-5981. DOI:
834 [10.1002/nme.2579](https://doi.org/10.1002/nme.2579). eprint: [https://onlinelibrary.wiley.com/doi/pdf/10.10](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2579)
835 [02/nme.2579](https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2579). URL: <https://doi.org/10.1002/nme.2579>.
- 836 [20] Carole Goble et al. “FAIR Computational Workflows”. In: *Data Intelligence* 2.1-2 (Jan.
837 2020), pp. 108–121. ISSN: 2641-435X. DOI: [10.1162/dint_a_00033](https://doi.org/10.1162/dint_a_00033). eprint: [https:](https://direct.mit.edu/dint/article-pdf/2/1-2/108/1893377/dint_a_00033.pdf)
838 [//direct.mit.edu/dint/article-pdf/2/1-2/108/1893377/dint_a_00033.pd](https://direct.mit.edu/dint/article-pdf/2/1-2/108/1893377/dint_a_00033.pdf)
839 [f](https://direct.mit.edu/dint/article-pdf/2/1-2/108/1893377/dint_a_00033.pdf). URL: https://doi.org/10.1162/dint_a_00033.
- 840 [21] Lars Griem et al. “KadiStudio: FAIR Modelling of Scientific Research Processes”. In:
841 *Data Science Journal* 21.1 (2022). DOI: [10.5334/dsj-2022-016](https://doi.org/10.5334/dsj-2022-016).
- 842 [22] Sebastiaan P. Huber. *aiida-shell*. Version 0.2.0. [https://github.com/sphuber/aiid](https://github.com/sphuber/aiida-a-shell)
843 [a-shell](https://github.com/sphuber/aiida-a-shell). June 2022.
- 844 [23] Sebastiaan P. Huber et al. “AiiDA 1.0, a scalable computational infrastructure for automated
845 reproducible workflows and data provenance”. In: *Sci Data* 7.1 (Sept. 2020). ISSN: 2052-
846 4463. DOI: [10.1038/s41597-020-00638-4](https://doi.org/10.1038/s41597-020-00638-4). URL: [1597-020-00638-4](https://doi.org/10.1038/s4
847 <a href=).

- 848 [24] Anubhav Jain et al. “FireWorks: A dynamic workflow system designed for high-throughput
849 applications”. In: *Concurrency Computat.: Pract. Exper.* 27.17 (May 2015), pp. 5037–
850 5059. ISSN: 1532-0626, 1532-0634. DOI: [10.1002/cpe.3505](https://doi.org/10.1002/cpe.3505). URL: <https://doi.org/10.1002/cpe.3505>.
- 852 [25] Ivo Jimenez et al. “The Popper Convention: Making Reproducible Systems Evaluation
853 Practical”. In: *2017 IEEE International Parallel and Distributed Processing Symposium
854 Workshops (IPDPSW)*. 2017, pp. 1561–1570. DOI: [10.1109/IPDPSW.2017.157](https://doi.org/10.1109/IPDPSW.2017.157).
- 855 [26] Farah Zaib Khan et al. “Sharing interoperable workflow provenance: A review of best
856 practices and their practical application in CWLProv”. In: *GigaScience* 8.11 (Nov. 2019).
857 ISSN: 2047-217X. DOI: [10.1093/gigascience/giz095](https://doi.org/10.1093/gigascience/giz095). URL: <https://doi.org/10.1093/gigascience/giz095>.
- 859 [27] Johannes Köster and Sven Rahmann. “Snakemake—a scalable bioinformatics workflow
860 engine”. In: *Method. Biochem. Anal.* 34.20 (May 2018), pp. 3600–3600. ISSN: 1367-4803,
861 1460-2059. DOI: [10.1093/bioinformatics/bty350](https://doi.org/10.1093/bioinformatics/bty350). URL: <https://doi.org/10.1093/bioinformatics/bty350>.
- 863 [28] Samuel Lampa et al. “SciPipe: A workflow library for agile development of complex
864 and dynamic bioinformatics pipelines”. In: *GigaScience* 8.5 (Apr. 2019). ISSN: 2047-
865 217X. DOI: [10.1093/gigascience/giz044](https://academic.oup.com/gigascience/article-pdf/8/5/giz044/28538382/giz044.pdf). eprint: <https://academic.oup.com/gigascience/article-pdf/8/5/giz044/28538382/giz044.pdf>. URL:
866 <https://doi.org/10.1093/gigascience/giz044>.
- 868 [29] Soohyun Lee et al. “Tibanna: Software for scalable execution of portable pipelines on the
869 cloud”. In: *Method. Biochem. Anal.* 35.21 (May 2019), pp. 4424–4426. ISSN: 1367-4803,
870 1460-2059. DOI: [10.1093/bioinformatics/btz379](https://academic.oup.com/bioinformatics/article-pdf/35/21/4424/31617561/btz379.pdf). eprint: <https://academic.oup.com/bioinformatics/article-pdf/35/21/4424/31617561/btz379.pdf>.
871 URL: <https://doi.org/10.1093/bioinformatics/btz379>.
- 873 [30] Felix Mölder et al. “Sustainable data analysis with Snakemake”. In: *F1000Res* 10 (Apr.
874 2021), p. 33. ISSN: 2046-1402. DOI: [10.12688/f1000research.29032.2](https://doi.org/10.12688/f1000research.29032.2). URL:
875 <https://doi.org/10.12688/f1000research.29032.2>.
- 876 [31] Barend Mons et al. “The FAIR Principles: First Generation Implementation Choices and
877 Challenges”. In: *Data Intellegence* 2.1-2 (Jan. 2020), pp. 1–9. ISSN: 2641-435X. DOI:
878 [10.1162/dint_e_00023](https://doi.org/10.1162/dint_e_00023). URL: https://doi.org/10.1162/dint_e_00023.
- 879 [32] Simon P. Sadedin, Bernard Pope, and Alicia Oshlack. “Bpipe: A tool for running and man-
880 aging bioinformatics pipelines”. In: *Method. Biochem. Anal.* 28.11 (Apr. 2012), pp. 1525–
881 1526. ISSN: 1460-2059, 1367-4803. DOI: [10.1093/bioinformatics/bts167](https://academic.oup.com/bioinformatics/article-pdf/28/11/1525/16905290/bts167.pdf). eprint:
882 <https://academic.oup.com/bioinformatics/article-pdf/28/11/1525/16905290/bts167.pdf>. URL: <https://doi.org/10.1093/bioinformatics/bts167>.
- 884 [33] Michael A. Salim et al. *Balsam: Automated Scheduling and Execution of Dynamic, Data-
885 Intensive HPC Workflows*. <https://arxiv.org/abs/1909.08704>. 2019. DOI:
886 [10.48550/ARXIV.1909.08704](https://arxiv.org/abs/1909.08704). URL: <https://arxiv.org/abs/1909.08704>.

- 887 [34] Joerg Schaarschmidt et al. “Workflow Engineering in Materials Design within the BAT-
888 TERY 2030+ Project”. In: *Advanced Energy Materials* 12.17 (2022), p. 2102638. DOI:
889 <https://doi.org/10.1002/aenm.202102638>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/aenm.202102638>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aenm.202102638>.
- 892 [35] Eduardo Naufel Schettino. *pydoit/doit: Task management & automation tool (python)*.
893 <https://doi.org/10.5281/zenodo.4892136>. June 2021. DOI: [10.5281/zenodo.4892136](https://doi.org/10.5281/zenodo.4892136). URL: <https://doi.org/10.5281/zenodo.4892136>.
- 895 [36] Nico Schlömer. *meshio: Tools for mesh files*. <https://doi.org/10.5281/zenodo.6346837>. Version v5.3.4. Mar. 2022. DOI: [10.5281/zenodo.6346837](https://doi.org/10.5281/zenodo.6346837). URL: <https://doi.org/10.5281/zenodo.6346837>.
- 898 [37] Will Schroeder et al. *The visualization toolkit : an object-oriented approach to 3D graphics*.
899 4th ed. Kitware, 2006.
- 900 [38] Michael Sperber et al. “Revised6 Report on the Algorithmic Language Scheme”. In: *J.*
901 *Funct. Program.* 19.S1 (Aug. 2009), p. 1. ISSN: 0956-7968, 1469-7653. DOI: [10.1017/s0956796809990074](https://doi.org/10.1017/s0956796809990074). URL: <https://doi.org/10.1017/s0956796809990074>.
- 903 [39] Martin Uhrin et al. “Workflows in AiiDA: Engineering a high-throughput, event-based
904 engine for robust and modular computational workflows”. In: *Nato. Sc. S. Ss. Iii. C. S.* 187
905 (Feb. 2021), p. 110086. ISSN: 0927-0256. DOI: [10.1016/j.commatsci.2020.110086](https://doi.org/10.1016/j.commatsci.2020.110086).
906 URL: <https://doi.org/10.1016/j.commatsci.2020.110086>.
- 907 [40] John Vivian et al. “Toil enables reproducible, open source, big biomedical data analyses”.
908 In: *Nat Biotechnol* 35.4 (Apr. 2017), pp. 314–316. ISSN: 1087-0156, 1546-1696. DOI:
909 [10.1038/nbt.3772](https://doi.org/10.1038/nbt.3772). URL: <https://doi.org/10.1038/nbt.3772>.
- 910 [41] Kate Voss, Geraldine Van Der Auwera, and Jeff Gentry. *Full-stack genomics pipelining*
911 *with GATK4 + WDL + Cromwell [version 1; not peer reviewed]*. slides. [https://f1000](https://f1000research.com/slides/6-1381)
912 [research.com/slides/6-1381](https://f1000research.com/slides/6-1381). 2017. DOI: [10.7490/f1000research.1114634.1](https://doi.org/10.7490/f1000research.1114634.1).
913 URL: <https://f1000research.com/slides/6-1381>.
- 914 [42] Mark D. Wilkinson et al. “The FAIR Guiding Principles for scientific data management
915 and stewardship”. In: *Sci Data* 3.1 (Mar. 2016). ISSN: 2052-4463. DOI: [10.1038/sdata](https://doi.org/10.1038/sdata.2016.18)
916 [.2016.18](https://doi.org/10.1038/sdata.2016.18). URL: <https://doi.org/10.1038/sdata.2016.18>.
- 917 [43] Peter Williams and Contributors. *The Tectonic Typesetting System*. [https://tectonic-](https://tectonic-typesetting.github.io/en-US/)
918 [typesetting.github.io/en-US/](https://tectonic-typesetting.github.io/en-US/). Accessed: 2022-06-02. 2022.
- 919 [44] Andy Wingo et al. *GNU Guile*. <https://www.gnu.org/software/guile/>. Feb. 2022.
- 920 [45] Laura Wratten, Andreas Wilm, and Jonathan Göke. “Reproducible, scalable, and shareable
921 analysis pipelines with bioinformatics workflow managers”. In: *Nat Methods* 18.10 (Sept.
922 2021), pp. 1161–1168. ISSN: 1548-7091, 1548-7105. DOI: [10.1038/s41592-021-012](https://doi.org/10.1038/s41592-021-01254-9)
923 [54-9](https://doi.org/10.1038/s41592-021-01254-9). URL: <https://doi.org/10.1038/s41592-021-01254-9>.
- 924 [46] Ricardo Wurmus et al. *GUIX Workflow Language*. <https://guixwl.org>. Version 0.5.0.
925 July 2022.

RESEARCH ARTICLE

- 926 [47] Andy B. Yoo, Morris A. Jette, and Mark Grondona. “SLURM: Simple Linux Utility for
927 Resource Management”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by
928 Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Berlin, Heidelberg: Springer
929 Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-39727-4.