DOI:10.1145/3486897

Standardizing computational reuse and portability with the Common Workflow Language.

BY MICHAEL R. CRUSOE, SANNE ABELN, ALEXANDRU IOSUP, PETER AMSTUTZ, JOHN CHILTON, NEBOJŠA TIJANIĆ, HERVÉ MÉNAGER, STIAN SOILAND-REYES, BOGDAN GAVRILOVIĆ, CAROLE GOBLE, AND THE CWL COMMUNITY

Methods Included

COMPUTATIONAL WORKFLOWS ARE widely used in data analysis, enabling innovation and decision-making for the modern society. But their growing popularity is also a cause for concern. Unless we standardize computational reuse and portability, the use of workflows may end up hampering collaboration. How can we enjoy the common benefits of computational workflows and eliminate such risks?

To answer this general question, in this work we advocate for workflow thinking as a shared method of reasoning across all domains and practitioners, introduce Common Workflow Language (CWL) as a pragmatic set of standards for describing and sharing computational workflows, and discuss the principles around which these standards have become central to a diverse community of users across multiple fields in science and engineering. This article focuses on an overview of CWL standards and the CWL project and is complemented by the technical detail available in the CWL standards.[a]

Workflow thinking is a form of "conceptualizing processes as recipes and protocols, structured as dataflow [or workflow] graphs with computational steps, and subsequently developing tools and approaches for formalizing, analyzing, and communicating these process descriptions."[14] It introduces the workflow, an abstraction which helps decouple expertise in a specific domain—for example, specific science or engineering fields—from computing expertise. Derived from workflow thinking, a *computational workflow* describes a process for computing where different parts of the process (the tasks) are interdependent—for instance, a task can start processing after its predecessors have (partially) completed and where data flows between tasks.

Currently, many competing systems exist to enable simple workflow execution (*workflow runners*) or offer comprehensive management of workflows and data (*workflow management systems*). Each has its own syntax or method for describing workflows and infrastructure requirements, which can limit computational reuse and portability. Although dataflows are becoming more complex, most workflow abstractions do not enable explicit specifications of dataflows, significantly increasing the cost to have third parties reuse and port the workflow.

a Common Workflow Language Standards, v1.2: https://w3id.org/cwl/v1.2/.

» key insights

- **Common Workflow Language is a set of open standards for describing and sharing computational workflows, used in many science and engineering domains.**
- **CWL standards support critical workflow concepts such as automation, scalability, abstraction, provenance, portability, and reusability.**
- **CWL standards are developed around core principles of community and shared decision making, reuse, and zero cost for participants.**

IMAGE BY ANDREW OSTROVSKY

We thus identify an important problem in the broad, practical adoption of workflow thinking: Although communities require *polylingual workflows* (workflows that execute tools written in multiple, different computer languages) and *multiparty workflows*, adopting and managing different workflow systems is costly and difficult. In this work, we propose to tame this complexity through a common abstraction that covers most features used in practice and that is (or can be) implemented in many workflow systems.

In the computational workflow depicted in Figure 1, practitioners solved the problem by adopting the CWL standards. In this work, we posit that the CWL standards provide the common abstraction that can help overcome the main obstacles to sharing workflows between institutions and users. CWL achieves this by providing a declarative language that allows expressing computational workflows constructed from diverse software tools—each executed through their command-line interface, with the inputs and outputs of each tool clearly specified and with inputs possibly resulting from the execution of other tools. We also set out to introduce the CWL standards, with a threefold focus:

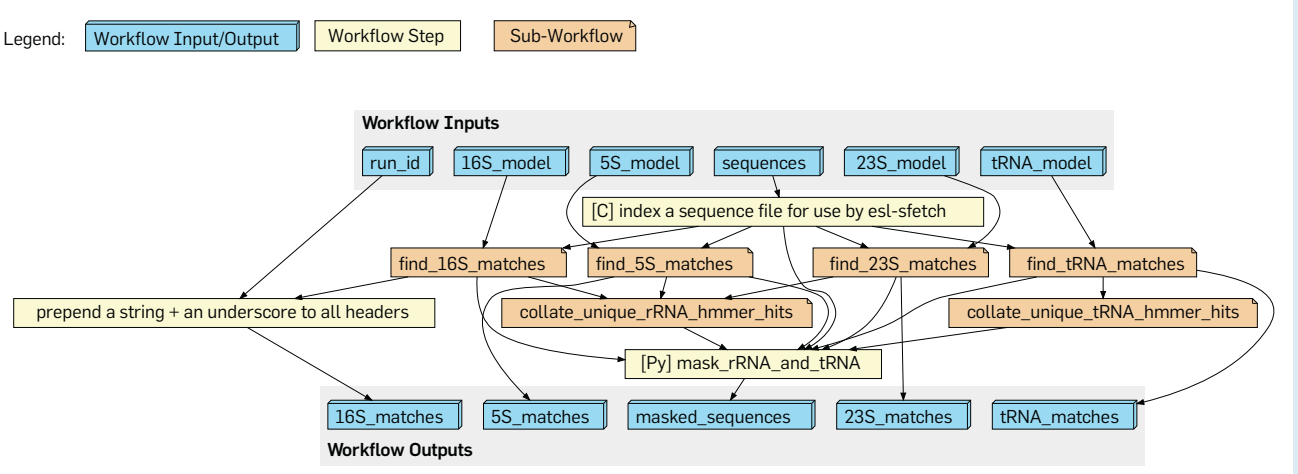
1. The CWL standards focus on maintaining a separation of concerns between the description and execution of tools and workflows, proposing a language that only includes operations commonly used across multiple communities of practice.
2. The CWL standards support workflow automation, scalability, abstraction, provenance, portability, and reusability.
3. To achieve these results, the CWL project takes a principled, community-first open source and open-standard approach.

The CWL standards are the product of an open and free standards-making community. While the CWL project began in bioinformatics, its many contributors shaped the standards to be useful in any domain that faces the problem of "many tools written in many programming languages by many parties." Since the ratification of the first version in 2016, the CWL standards have been used in other fields, including hydrology, radio astronomy, geo-spatial analysis,[13,23,32] and high-energy physics,[4] in addition to fast-growing bioinformatics fields such as metagenomics[27] and cancer research.[24] The CWL standards are featured in the IEEE 2791-2020 standard, sponsored and adopted by the U.S. FDA,[16] and the Netherlands' National Plan for Open Science.[34] A list of free and open source implementations of the CWL standards is offered in the Table. Multiple, commercially supported systems that follow the CWL standards for executing workflows are also available from vendors such as Curii (Arvados), DNAnexus, IBM (IBM® Spectrum LSF), Illumina (Illumina Connected Analytics), and Seven Bridges. The flexibility of the CWL standards enabled, for example, rapid collaboration on and prototyping of a COVID-19 public database and analysis resource.[15]

The separation of concerns proposed by the CWL standards enables diverse projects and can also benefit engineering and large industrial projects. Likewise, users of Docker or other software-container technologies that distribute analysis tools can leverage just the CWL Command Line Tool

Figure 1. Excerpt from a large microbiome bioinformatics CWL workflow.[27,*]

This part of the workflow, which is interpretable/executable on its own, aims to match the workflow inputs of genomic sequences to provided sequence models, which are dispatched to four sub-workflows (for instance, find_16S_matches). The sub-workflows are not detailed in the figure. Sub-workflow outputs are collated to identify unique sequence hits and then provided as overall workflow outputs. Arrows define the connection between tasks and imply their partial ordering, depicted here as layers of tasks that may execute concurrently. Workflow steps—for example, "mask_rRNA_and_tRNA"—execute command-line tools, shown here with indicators for their different programming languages ([Py] for Python, [C] for the C language).



* Diagram adapted from https://w3id.org/cwl/view/git/7bb76f33bf40b5cd2604001cac46f967a209c47f/workflows/rna-selector.cwl, which was originally retrieved from a corresponding CWL workflow of the EBI Metagenomics project, itself a conversion of the "rRNASelector"[25] program into a well-structured workflow, allowing for better parallelization of execution and provenance tracking.

standard to access a structured, workflow-independent description of how to run their tool(s) in the container, what data must be provided to the container, expected results, and where to find them.

Background on Workflows and Standards for Workflows

Workflows, and standards-based descriptions thereof, hold the potential to solve key problems in many domains of science and engineering.

Why workflows? In many domains, workflows include diverse analysis components, written in multiple, different computer languages by both end users and third parties. Such polylingual and multi-party workflows are already common or dominant in data-intensive fields, such as bioinformatics, image analysis, and radio astronomy. We envision they could bring important benefits to many other domains.

To thread data through analysis tools, domain experts such as bioinformaticians use specialized command-line interfaces,[12,31] while experts in other domains use proprietary, customized frameworks.[2,5] Workflow engines also help with efficiently managing the resources used to run scientific workloads.[7,10]

The workflow approach helps compose an entire application of these command-line analysis tools: Developers build graphical or textual descriptions of how to run these command-line tools, and scientists and engineers connect their inputs and outputs so that the data flows through. An example of a complex workflow problem is metagenomic analysis, for which Figure 1 illustrates a subset (a *sub-workflow*).

In practice, many research and engineering groups use workflows of the kind described in Figure 1. However, as highlighted in a "Technology Toolbox" article recently published in *Nature*,[29] these groups typically lack the ability to share and collaborate across institutions and infrastructures without costly manual translation of their workflows.

Using workflow techniques, especially with digital analysis processes, has become quite popular and does not appear to be slowing down. One workflow-management system, Galaxy Publication Library, recently celebrated its 10,000th citation, and more than 309 computational data-analysis workflow systems are known to exist.[b] A process, digital or otherwise, may grow to such complexity that its authors and users have difficulties understanding its structure, scaling and managing it, and keeping track of what happened in the past. Process dependencies may be undocumented, obfuscated, or otherwise effectively invisible. Outsiders or newcomers may find even an extensively documented process difficult to understand if it lacks a common framework or vocabulary. The need to run the process more frequently or with larger inputs is unlikely to be achieved by the initial entity—that is, either a script or a human—running the process. What seemed once a reasonable manual step—run this command here, paste the result there, and then call this person for permission—will become a bottleneck under the pressure of porting and reusing. Informal logs (if any) will quickly become unsuitable for helping an organization understand what happened, when, by whom, and to which data.

b Existing Workflow Systems: https://s.apache.org/existing-workflow-systems.

Monolingual and Polylingual Workflow Systems

Techniques for workflows can be implemented in many ways—that is, with varying degrees of formalism—which tends to correlate with execution flexibility and features. Whereas the most informal techniques typically require that all processing components are written in or are at least callable from the same programming language, formal workflow techniques tend to allow components to be developed in multiple programming languages.

Among the informal techniques, the do-it-yourself approach uses built-in capabilities from a particular programming language. For example, Python provides a *threading library*, and the Java-based Apache Hadoop[33] provides MapReduce capabilities. To gain flexibility when working with a particular programming language, general third-party libraries, such as *ipyparallel*,[a] can enable remote or distributed execution without having to rewrite one's code.

A more explicit workflow structure can be achieved by using a *workflow library* focusing on a specific programming language. For example, in Parsl,[2] the workflow constructs ("this is a unit of processing" or "here are the dependencies between the units") are made explicit and added by the developer to a Python script, to upgrade it to a scalable workflow. (While we list Parsl as an example of a monolingual workflow system, it also contains explicit support for executing external command-line tools.)

Two approaches—the use of per-language *add-in libraries* or the use of the *Portable Operating System Interface command-line interface (POSIX CLI)*[30]—can accommodate polylingual workflows, where components are written in more than one programming language or where components come from third parties and the user does not want to or cannot modify them. Using per-language add-in libraries entails either explicit function calls (for example, using Python *ctypes* to call a C library[b]) or the addition of annotations to the user's functions; this requires mapping/restricting to a common, cross-language data model.

Essentially all programming languages support the creation of POSIX CLIs, which are familiar to many Linux and macOS users as scripts or binaries that can be invoked on the shell with a set of arguments, reading and writing files, and executed in a separate process. Choosing the POSIX command-line interface as the coordination point means the connection between components is performed by an array of string arguments representing program options (including paths to data files) along with string-based environment variables (key-value pairs). Using the command line as a coordination interface has the advantage of not needing additional implementation in every programming language but is challenged by process start-up time and a very simple data model. (As a polylingual workflow standard, CWL uses the POSIX CLI data model.)

a IPython Parallel (ipyparallel) is a Python package and collection of CLI scripts for controlling clusters of IPython processes, built on the Jupyter protocol. See https://pypi.org/project/ipyparallel/.

b ctypes is a foreign function library for Python. See https://docs.python.org/3/library/ctypes.html.

Selected F/OSS workflow runners and platforms that implement the CWL standards.

Implementation	Platform Support
cwltool	Linux, macOS, MS Windows (via WSL 2) local execution only
Arvados	In the cloud on AWS, Azure, and Google Cloud Platform (GCP), on-premise and hybrid clusters using Slurm or LSF
Toil[35]	AWS, Azure, GCP, Grid Engine, HTCondor, IBM Spectrum LSF, Mesos, OpenStack, Slurm, PBS/Torque; also local execution on Linux, macOS, and MS Windows (via WSL 2)
CWL-Airflow[21]	Local execution on Linux, OS X, or via dedicated Airflow-enabled cluster.
StreamFlow[6]	Kubernetes, HPC with Singularity (PBS, Slurm), Occam, multi-node SSH, and local-only (Docker, Singularity)
REANA	Kubernetes

Workflow techniques aim to solve these problems by providing the abstraction, scaling, automation, and provenance (ASAP) features.[8] Workflow constructs enable a clear abstraction about the components, the relationships between them, and the inputs and outputs of the components turning them into well-labeled tools with documented expectations. This abstraction enables:

- **Scaling:** Execution can be parallelized and distributed.
- **Automation:** The abstraction can be used by a workflow engine to track, plan, and manage task execution.
- **Provenance tracking:** Descriptions of tasks, executors, inputs, and outputs—with timestamps, identifiers (unique names), and other logs—can be stored in relation to each other to later answer structured queries.

Why workflow standards? Although workflows are very popular, prior to the CWL standards, all workflow systems were incompatible with each other. This means that users who do not use the CWL standards are required to express their computational workflows in a different way each time they use another workflow system, leading to local success but global unportability.

The success of workflows is now their biggest drawback. Users are locked into a particular vendor, project, and often a specific hardware setup, hampering sharing and reuse. Even non-academics suffer from this situation, as the lack of standards, or their adoption, hinders effective collaboration on computational methods within and between companies. Likewise, this unportability affects public/private partnerships and the potential for technology transfer from public researchers.

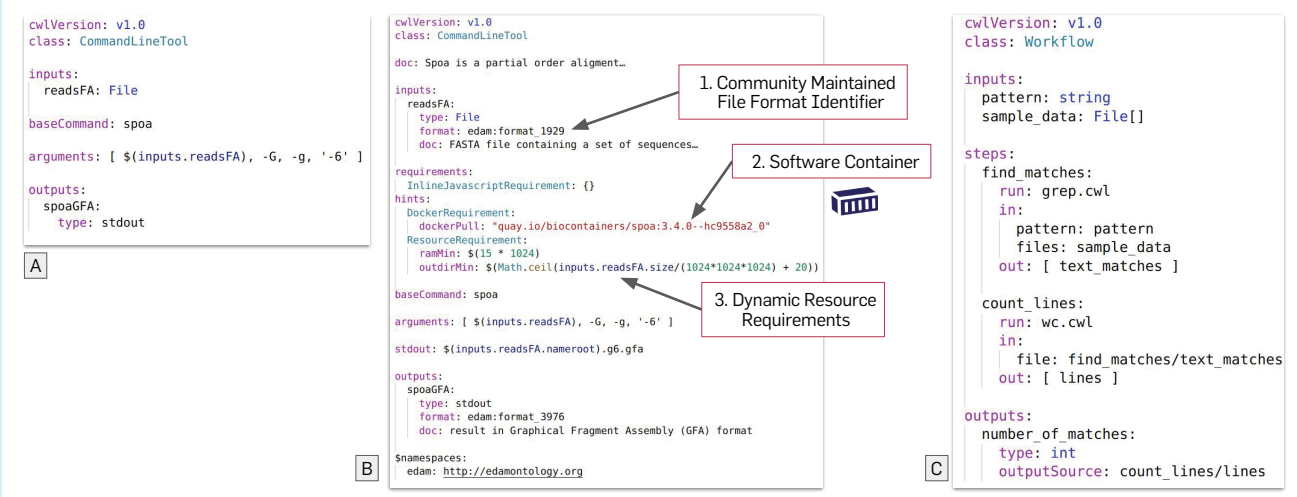
A second significant problem is that incomplete method descriptions are common when computational analysis is reported in academic research.[17] Reproduction, reuse, and replication[11] of these digital methods requires a complete description of which computer applications were used, how they were used, and how they were connected to each other. For precision and interoperability, this description should also be in an appropriate, standardized, machine-readable format.

A standard for sharing and reusing workflows can provide a solution to describing portable, reusable workflows while also being workflow-engine and vendor neutral.

Sharing workflow descriptions based on standards also addresses the second problem: The availability of the workflow description provides needed information when sharing, and the quality of the description provided by a structured, standards-based approach is much higher than the current approach of casual, unstructured, and almost always incomplete descriptions in scientific reports. Moreover, the operational parts of the

Figure 2. Example of CWL syntax and progressive enhancement.

(a) and (b) describe the same tool, but (b) is enhanced with additional features: human-readable documentation; file format identifiers for better validation of workflow connections; recommended software container image for more reproducible results and easier installation; and dynamically specified resource requirements to optimize task scheduling and resource usage without manual intervention. Resource requirements are expressed as *hints*. (c) shows an example of CWL Workflow syntax, where the underlying tool descriptions ("grep.cwl" and "wc.cwl") are in external files for ease of reuse.



description can be automated by the workflow-management system rather than by domain experts.

While (data) standards are commonly adopted and have become expected for funded projects in knowledge representation fields, the same cannot yet be said about workflows and workflow engines.

Features of the Common Workflow Language Standards

The Common Workflow Language standards aim to cover the common needs of users and the commonly implemented features of workflow runners or platforms. The remainder of this section presents an overview of CWL features, how they translate to executing workflows in CWL format, and where the CWL standards are not helpful.

The CWL standards support polylingual and multi-party workflows, for which they enable computational reuse and portability. To do so, each release of the CWL standards has two[c] main components: (1) a standard for describing command-line tools and (2) a standard for describing workflows that compose such tool descriptions. The goal of the *CWL Command Line Tool Description Standard* is to describe how a particular command-line tool works: What are the inputs and parameters and their types? How do you add the correct flags and switches to the command-line invocation? Where do you find the output files?

The CWL standards define an *explicit language*, both in syntax and in its data and execution model. Its textual syntax, derived from YAML,[d] does not restrict the amount of detail. For example, Figure 2a depicts a simple example with sparse detail, and Figure2b depicts the same example but with the execution augmented with more details. Each input to a tool has a name and a type—for instance, File (see Figure 2b, Item 1). Tool-description authors are encouraged to include documentation and labels for all components (as shown in Figure 2b), to enable the automatic generation of helpful visual depictions and even graphical user interfaces (GUIs) for any given CWL description. Metadata about the tool-description authors encourages attribution of their efforts. As shown in Figure 2b, Item 3, these tool descriptions can contain well-defined hints or mandatory requirements, such as which software container to use or the amount of required compute resources: memory, number of CPU cores, amount of disk space, and/or the maximum time or deadline to complete the step or entire workflow.

The CWL execution model is explicit. Each tool's runtime environment is explicit, and any required elements must be specified by the CWL tool-description author (in contrast to hints, which are optional).[e] Each tool invocation uses a separate working directory, populated according to the CWL tool description—for example, with the input files explicitly specified by the workflow author. Some applications expect particular filenames, directory layouts, and environment variables, and there are additional constructs in the CWL Command Line Tool standard to satisfy their needs.

The explicit runtime model enables portability, by being explicit about data locations. As Figure 3 indicates, this enables the execution of CWL workflows in diverse environments as provided by various implementations of the CWL standards: the local environment of the author-scientist (for instance, a single desktop computer, laptop, or workstation), a remote batch production environment (for example, a cluster, an entire data center, or even a global multi-data center infrastructure), and an on-demand cloud environment.

The CWL standards explicitly support the use of *software container* technologies, such as Docker and Singularity, to enable the portability of the underlying analysis tools. Figure 2b, Item 2 illustrates the process of pulling a Docker container image from the Quay.io registry; then, the workflow engine automates the mounting of files and folders within the container. The container included in the figure has been developed by a trusted author and is commonly used in the bioinformatics field, with the expectation that its results are reproducible. Indeed, the use of containers can be seen as a confirmation that a tool's execution is reproducible when using only its explicitly declared runtime environment. Similarly, when *distributed execution* is desired, no changes to the CWL tool description are needed. File or directory inputs are already explicitly defined in the CWL description, so the (distributed) workflow runner can handle job placement and data routing between compute nodes without additional configuration.

Via these two features—special han-

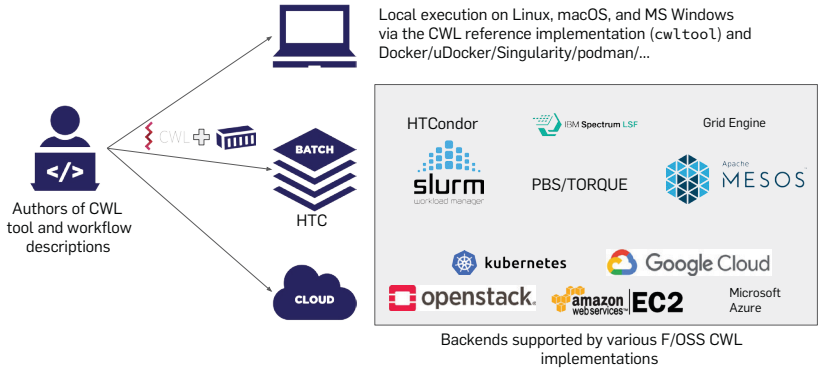
c The third component, Schema Salad, is only of interest to those who want to parse the syntax of the schema language that is used to define the syntax of CWL itself.

d JSON is an acceptable subset of YAML, and common when converting from another format to CWL syntax.

e For details on how the CWL Command Line Tool standard specifies that tool executors should set up and control the runtime environment, visit: https://w3id.org/cwl/v1.2/CommandLineTool.html#Runtime_environment, which also specifies which directories tools are allowed to write to.

Figure 3. Example of CWL portability.

The same workflow description runs on the scientist's own laptop or single machine, on any batch-production environment, and on any common public or private cloud. The CWL standards enable execution portability by being explicit about data locations and execution models.



The CWL Project and Free/Open Source Software (F/OSS)

Free and Open Source implementations of the CWL standards. **As of 2021, the CWL standards have gained much traction and are widely supported in practice. In addition to the implementations in the Table, Galaxy[1] and Pegasus[10] also have in-development support for the CWL standards.**

Wide adoption benefits from our principles: The CWL standards include conformance tests, but the CWL community does not yet test or certify implementations of the standards or specific technology stacks. Instead, the authors and service providers of workflow runners and workflow-management systems self-certify support for the CWL standards, based on a particular technology configuration they deploy and maintain.

F/OSS tools and libraries for working with CWL-format documents.[a] **CWL plug-ins exist for Atom, Vim, Emacs, Visual Studio Code, IntelliJ, gedit, and any editor that supports the Language Server Protocol (LSP)[b] standard. There are tools to generate CWL syntax from Python (via argparse/click or via functions), ACD,[c] CTD,[d] and annotations in IPython Jupyter Notebooks. Libraries to generate and/or read CWL documents exist in many languages: Python, Java, R, Go, Scala, Javascript, Typescript, and C++.**

a Summarized from https://www.commonwl.org/tools/.
b https://microsoft.github.io/language-server-protocol/.
c "Ajax Command Definitions" as produced by the EMBOSS tools: http://emboss.sourceforge.net/developers/acd/.
d XML-based "Common Tool Descriptors"[9] originating in the OpenMS project: https://github.com/WorkflowConversion/CTDSchema.

dling of data paths and the optional but recommended use of software containers—the CWL standards enable portability (execution "without change"). While portability can be affected by various factors not controllable by software container technology—for instance, variation in the underlying operating-system kernel or in processor results—in practice, the exact same software container and data inputs lead to portability without further adjustment from the user.

To support features that are not in the CWL standards, the standards define *extension points* that permit namespaced, vendor-specific features in explicitly defined ways. If these extensions do not fundamentally change how the tool should operate, then they are added to the hints list, and other CWL-compatible engines can ignore them. However, if the extension is required to properly run the tool being described—for instance, due to the need for some specialized hardware—then the extension is listed under requirements, and CWL-compatible engines can recognize and explicitly declare their inability to execute that CWL description.

The CWL Workflow Description Standard builds upon the CWL Command Line Tool Standard. It has the same YAML- or JSON-style syntax, with explicit workflow-level inputs, outputs, and documentation (Figure 2c). Workflow descriptions consist of a list of steps, comprising CWL Command Line Tools or CWL sub-workflows, each re-exposing their tool's required inputs. Inputs for each step are connected by referencing the name of either the common *workflow inputs* or of outputs from other steps. The *workflow outputs* expose selected outputs from workflow steps, making explicit which intermediate-step outputs will be returned from the workflow. All connections include identifiers, which CWL document authors are encouraged to name meaningfully—for example, "reference_genome" instead of "input7."

CWL workflows form explicit dataflows, as required for a particular computational analysis. The connectivity between steps defines the partial execution order. Parallel execution of steps is permitted and encouraged whenever multiple steps have all their inputs satisfied. For example, in Figure 1, "find_16S_matches" and "find_S5_matches" are at the same data-dependency level and can execute concurrently or sequentially in any order. Additionally, a *scatter* construct allows the repeated execution of a CWL step (perhaps overlapping in time, depending on the available resources), where most of the inputs are the same except for one or more inputs that vary. This is done without having to modify the underlying tool description. Starting with CWL version 1.2, workflows can also conditionally skip execution of a step (tool or workflow), based upon a specified intermediate input or custom Boolean evaluation. Combining these features allows for a flexible *branch* mechanism, which allows workflow engines to calculate data dependencies before the workflow starts and thus retains the predictability of the dataflow paradigm.

In contrast to hard-coded approaches that rely on implicit file paths specific to each workflow, CWL workflows are more flexible, reusable, and portable, which enables scalability. The use of explicit runtime environments in the CWL standards, combined with explicit inputs/outputs to form the dataflow, enables step reordering and explicit handling of iterations. The same features enable scalable remote execution and, more generally, flexible use of runtime environments. Moreover, individual tool definitions from multiple workflows can be reused in any new workflow.

CWL workflow descriptions are also future proof. Forward compatibility of CWL documents is guaranteed, as each CWL document declares which version of the standards it was written for, and minor versions do not alter the required features of the major version. A standalone upgrader can automatically upgrade CWL documents from one version to the next, and many CWL-aware platforms will internally update user-submitted documents at runtime.

Execution of workflows in CWL format. CWL is a set of standards, not a particular software product to install, purchase, or rent. The CWL standards need to be implemented to be useful; a list of some implementations of the CWL standards is in the Table. Workflow/tool runners that claim compliance with the CWL standards are allowed significant flexibility in how and where they execute a user's CWL documents as long as they fulfill the require-

ments written in those documents. For example, they are allowed (and encouraged) to distribute execution of a workflow across all available computers that can fulfill user-specified resource requirements. Aspects of execution not defined by the CWL standards include Web APIs for workflow execution and real-time monitoring.

For example, details about when a step should be considered ready for execution are available in Section 4 of the CWL Workflow Description standard, but once all the inputs are available, the exact timing is up to the workflow engine itself.

Step execution may result in a temporary or permanent failure, as defined in Section 4 of the CWL Workflow Description standard. The workflow engine must control any automatic failure recovery attempts—for instance, to re-execute a workflow step. Most workflow engines that implement the CWL standards feature the ability to attempt several re-executions, set by the user, before reporting permanent failure.

The CWL community has developed the following optimizations without requiring that users rewrite their workflows to benefit:

- Automatic streaming of data inputs and outputs instead of waiting for all data to be downloaded or uploaded (where those data inputs or outputs are marked with "streamable: true").
- Workflow step placement based on data location,[18] resource needs, and/or cost of data transfer.[19]
- The reuse of the results from previously computed steps, even from a different workflow, as long as the inputs are identical. This can be controlled by the user via the "WorkReuse" directive in the CWL Workflow Standard.

Real-world usage at scale. CWL users and vendors routinely report that they analyze 5,000 whole-genome sequences in a single workflow execution. One customer of a commercial vendor reported a successful workflow run containing an 8,000-wide step; the entire workflow had 25,000 container executions. By design, the CWL standards do not impose any technical limitations on the size of files processed or to the number of tasks run in parallel. The major scalability bottlenecks are hardware-related—not having enough machines with enough memory, compute power, or disk space to process ever-growing data at a greater scale. As these boundaries move in the future with technological advances, the CWL standards should be able to keep up and not be a limitation.

When is CWL not useful? The CWL standards were designed for a particular style of command-line, tool-based data analysis. Therefore, the following situations are out of scope and not appropriate (or possible) to describe using CWL syntax:

- Safe interaction with stateful (web) services.
- Real-time communication between workflow steps.
- Interactions with command-line tools beside 1) constructing the command line and making available file inputs (both user-provided and synthesized from other inputs just prior to execution) and 2) consuming the output of the tool once its execution is finished, in the form of files created/changed, the POSIX standard output and error streams, and the POSIX exit code of the tool.
- Advanced control-flow techniques beyond conditional steps.
- Runtime workflow graph manipulations: dynamically adding or removing new steps during workflow execution, beyond any predefined conditional step execution tests that are in the original workflow description.
- Workflows that contain cycles: "Repeat this step or sub-workflow a specific number of times" or "Repeat this step or sub-workflow until a condition is met."[f]
- Workflows that need specific steps run on a specific day or at a specific time.

Open Source, Open Standards, Open Community

Given the numerous and diverse set of potential users, implementers, and other stakeholders, we posit that a project like CWL requires the combined development of code, standards, and community. Indeed, these requirements were part of the foundational design principles for CWL; in the long run, these principles have fostered free and open source software (see sidebar "The CWL Project and Free/Open Source Software") and a vibrant and active ecosystem.

The CWL principles. The CWL project is based on a set of five principles:

- **Principle 1:** At the core of the project is the community of people who care about its goals.
- **Principle 2:** To achieve the best possible results, there should be few, if any, barriers to participation. Specifically, to attract people with diverse experiences and perspectives, there must be no cost to participate.
- **Principle 3:** To enable the best outcomes, project outputs should be used as people see fit. Thus, the standards themselves must be licensed for reuse, with no acquisition price.
- **Principle 4:** The project must not favor any one company or group over another, but neither should it try to be all things to all people. The community decides.
- **Principle 5:** Concepts and ideas must be tested frequently. Tested and functional code is the beginning of evaluating a proposal, not the end.

Over time, CWL project members learned that this approach is a superset of the OpenStand Principles, a joint "Modern Paradigm for Standards" promoted by the IAB, IEEE, IETF, Internet Society, and W3C. The CWL project additions to the OpenStand Principles are 1) to keep participation free of cost, and 2) the explicit choice of Apache License 2.0 for all its text, conformance tests, and reference implementations.

Necessary and sufficient. All these principles have proven to be essential for the CWL project. For example, Principles 2 and 3 have enabled many implementations of the CWL standards, several of which reuse different parts of the reference implementation of the CWL standards (*reference runner*). Being community-first, per Principle 1, has led participants to create several projects that are outside the CWL standards; the most important contributions have made their way back into the project (Principle 4).

As part of Principle 5, contributors to the CWL project have developed a

f Supporting cycles/loops as an optional feature has been suggested for a future version of the CWL standards, but it has yet to be put forth as a formal proposal with a prototype implementation. As a work around, one can launch a CWL workflow from within a workflow system that does support cycles, as documented in the eWaterCycle case study with Cylc.[28]

suite of conformance tests for each version of the CWL standards. These publicly available tests were critical to the CWL project's success: They helped assess the reference implementation of the CWL standards, they provided early adopters with concrete examples, and they enabled developers and users of production implementations of the CWL standards to confirm their accuracy.

The CWL ecosystem. Beyond the ratified initial and updated CWL standards released over the last six years, the CWL community has developed many tools, software libraries, and connected specifications, and has shared CWL descriptions for popular tools. For example, there are software development kits (SDKs) for both Python and Java that are generated automatically from the CWL schema. This allows programmers to load, modify, and save CWL documents using an object-oriented model that directly corresponds to the standards themselves. CWL SDKs for other languages are possible by extending the code generation routines.[g] (See Sidebar: The CWL Project and Free/Open Source Software for practical details.)

The CWL standards offer strong support for the acute need to reuse (and, correspondingly, to share) information on workflow execution as well as on authoring and provenance. The *CWLPROV* prototype was created to show how existing standards[3,22,26] can be combined to represent the provenance of a specific execution of a CWL workflow.[20] Although, to date, CWLProv has only been implemented in the CWL reference runner, interest in further development and implementation is high.

Conclusion

The problem of standardizing computational reuse is only increasing in prominence and impact. Addressing this problem, various domains in science, engineering, and commerce have already started migrating to workflows, but efforts focusing on the portability and even definition of workflows remain scattered. In this work, we raise awareness to this problem and propose a community-driven solution.

By design, the CWL standards do not impose any technical limitations on the size of files processed or to the number of tasks run in parallel.

The CWL is a family of standards for the description of command-line tools and of the workflows made from these tools. It includes many features developed in collaboration with the community: support for software containers, resource requirements, workflow-level conditional branching, and more. Built on a foundation of five guiding principles, the CWL project delivers open standards, open source code, and an open community.

For the past six years, the CWL community has grown organically. Organizations looking to write, use, or fund data-analysis workflows based upon command-line tools should adopt or even require the CWL standards, because they offer a common yet reduced set of capabilities that are both used in practice and implemented in many popular workflow systems. There are other ways CWL offers value: It is supported by a large-scale community, diverse fields have already adopted it, and its adoption is rapidly growing. Specifically:

1. With a reduced set of capabilities, the CWL standards limit the complexity encountered by new users when they first start and by operators during implementation. Feedback from the community indicates these are appreciated.

2. CWL's use of declarative syntax allows users to specify workflows even if they do not know exactly where the workflows would (later) run.

3. The CWL project is governed in the public interest and produces freely available open standards. The CWL project itself is not a specific workflow-management system, workflow runner, or vendor. This allows potential users, operators, and vendors to avoid lock-in and be more flexible in the future.

4. By offering standards, the CWL project distinguishes itself, especially for the complex interactions that appear in scientific and engineering collaborations. These interactions include defining workflows from many different tools (or steps), sharing workflows, long-term archiving, fulfilling requirements of regulators (for example, U.S. FDA), and making workflow executions auditable and reproducible. This is especially useful in cooperative environments, where groups that compete

g See the *codegen*.py files in https://pypi.org/project/schema-salad/7.1.20210316164414/.

also need to collaborate, or in scientific papers where results can be reused very efficiently if the analysis is described in a CWL workflow with publicly available software containers for all steps.

5. The CWL standards are already implemented, adopted, and used, with many production-grade implementations available as open source and with zero-cost. Thus, the different communities of users of the CWL standards already offer numerous workflow and tool descriptions. This is akin to how the Python ecosystem of shared libraries, code, and recipes is already helpful.

This is a call for others to embrace workflow thinking and join the CWL community in creating and sharing portable and complete workflow descriptions. With the CWL standards, the methods are included and ready to (re)use.

Acknowledgments

The list of those involved in the CWL project we would like to acknowledge is extensive. A comprehensive list is available in the online supplementary material: https://bit.ly/3MoOPfQ. **Funding acknowledgments:** European Commission grants BioExcel-2 (SSR) H2020-INFRAEDI-02-2018 823830, BioExcel (SSR) H2020-EINFRA-2015-1 675728, EOSCLife (SSR) H2020-INFRAEOSC-2018-2 824087, EOSCPilot (MRC) H2020-INFRADEV-2016-2 739563, IBISBA 1.0 (SSR) H2020-INFRAIA-2017-1-two-stage 730976, ELIXIR-EXCELERATE (SSR, HM) H2020-INFRADEV-1-2015-1 676559, ASTERICS (MRC) INFRADEV-4-2014-2015. ELIXIR the research infrastructure for life-science data, Interoperability Platform Implementation Study (MRC). 2018-CWL. Various universities have also co-sponsored this project. We thank Vrije Universiteit of Amsterdam, the Netherlands, where the first three authors have their primary affiliation.

References

1. Afgan, E. et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research 46*, W1 (July 2018), W537–W544. https://doi.org/10.1093/nar/gky379.
2. Babuji, Y. et al. Parsl: Pervasive parallel programming in Python. In *Proceedings of the 28th Intern. Symp. on High-Performance Parallel and Distributed Computing*. Association for Computing Machinery (2019), 25–36. https://doi.org/10.1145/3307681.3325400.
3. Belhajjame, K. et al. Using a suite of ontologies for preserving workflow-centric research objects. *J. of Web Semantics 32* (May 2015), 16–42. https://doi.org/10.1016/j.websem.2015.01.003.
4. Bell, T. et al. *Web-based Analysis Services Report.* Technical Report CERN-IT-Note-2018-004. (2017), CERN, Geneva, Switzerland. http://cds.cern.ch/record/2315331/.
5. Berthold, M.R et al. KNIME–The Konstanz information miner: Version 2.0 and beyond. *ACM SIGKDD Explorations Newsletter 11*, 1 (Nov. 2009), 26–31. https://doi.org/10.1145/1656274.1656280.
6. Colonnelli, I. et al. StreamFlow: Cross-breeding cloud with HPC. *IEEE Transactions on Emerging Topics in Computing* (2020), 1–1. https://doi.org/10.1109/TETC.2020.3019202.
7. Couvares, P. et al. Workflow management in Condor. In *Workflows for e-Science: Scientific Workflows for Grids*, I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields (Eds.). Springer, London (2007), 357–375. https://doi.org/10.1007/978-1-84628-757-2_22.
8. Cuevas-Vicenttín, C. et al. Scientific workflows and provenance: Introduction and research opportunities. *Datenbank-Spektrum 12*, 3 (Nov. 2012), 193–203. https://doi.org/10.1007/s13222-012-0100-z.
9. de la Garza, L. et al. From the desktop to the grid: Scalable bioinformatics via workflow conversion. *BMC Bioinformatics 17*, 1 (March 2016), 127. https://doi.org/10.1186/s12859-016-0978-9.
10. Deelman, E. et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems 46* (May 2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008
11. Feitelson, D.G. From repeatability to reproducibility and corroboration. *ACM SIGOPS Operating Systems Review 49*, 1 (Jan. 2015), 3–11. https://doi.org/10.1145/2723872.2723875.
12. Georgeson, P. et al. Bionitio: Demonstrating and facilitating best practices for bioinformatics command-line software. *GigaScience 8*, giz109 (Sept. 2019). https://doi.org/10.1093/gigascience/giz109.
13. Gonçalves, P. OGC Earth observations applications pilot: Terradue engineering report. OGC Public Engineering Report OGC 20-042. Open Geospatial Consortium. http://docs.opengeospatial.org/per/20-042.html.
14. Gryk, M.R. and Ludäscher, B. Workflows and provenance: Toward information science solutions for the natural sciences. *Library Trends 65*, 4 (2017), 555–562. https://doi.org/10.1353/lib.2017.0018.
15. Guarracino, A. et al. COVID-19 PubSeq: Public SARS-CoV-2 sequence resource. Bioinformatics Open Source Conference (July 2020), https://sched.co/coLw.
16. IEEE standard for bioinformatics analyses generated by high-throughput sequencing (HTS) to facilitate communication. (May 11, 2020), https://doi.org/10.1109/IEEESTD.2020.9094416.
17. Ivie, P. and Thain, D. Reproducibility in scientific computing. *ACM Computing Surveys 51*, 3 (July 2018), 63:1–63:36. https://doi.org/10.1145/3186266.
18. Jiang, F. Castillo, C., and Ahalt, S. TR-19-01: A cloud-agnostic framework for geo-distributed data-intensive applications. RENCI, University of North Carolina at Chapel Hill, (2019), https://renci.org/technical-reports/tr-19-01/.
19. Jiang, F., Ferriter, K., and Castillo, C. PIVOT: Cost-aware scheduling of data-intensive applications in a cloud-agnostic system. RENCI, University of North Carolina at Chapel Hill, (2019). https://renci.org/technical-reports/tr-19-02/.
20. Khan, F.Z. et al. Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv. *GigaScience 8*, 11 (November 2019), giz095. https://doi.org/10.1093/gigascience/giz095.
21. Kotliar, M., Kartashov, A.V., and Barski, A. CWL-Airflow: A lightweight pipeline manager supporting Common Workflow Language. *GigaScience 8*, 7 (July 2019), giz095. https://doi.org/10.1093/gigascience/giz084.
22. Kunze, J., Littman, J., Madden, E., Scancella, J., and Adams, C. The BagIt file packaging format (V1.0). (October 2018), DOI 10.17487/RFC8493. https://www.rfc-editor.org/info/rfc8493.
23. Landry, T. OGC Earth observation applications pilot: CRIM engineering report. Open Geospatial Consortium Public Engineering Report 20-045 (2020), http://docs.opengeospatial.org/per/20-045.html
24. Lau, J.W. et al. The Cancer Genomics Cloud: Collaborative, reproducible, and democratized—A new paradigm in large-scale computational research. *Cancer Research 77*, 21 (Oct. 2017), e3–e6. https://doi.org/10.1158/0008-5472.can-17-0387.
25. Lee, J-H., Yi, H., and Chun, J. rRNASelector: A computer program for selecting ribosomal RNA encoding sequences from metagenomic and metatranscriptomic shotgun libraries. *The J. of Microbiology 49*, 4 (September 2011), 689. https://doi.org/10.1007/s12275-011-1213-z.
26. Missier, P., Belhajjame, K., and Cheney, J. The W3C PROV family of specifications for modelling provenance metadata. In *Proceedings of the 16th Intern. Conf. on Extending Database Technology*. Association for Computing Machinery (2013). https://doi.org/10.1145/2452376.2452478.
27. Mitchell, A-L. MGnify: The microbiome analysis resource in 2020. *Nucleic Acids Research 48*, D1 (January 2020), D570–D578. https://doi.org/10.1093/nar/gkz1035.
28. Oliver, H. Workflow automation for cycling systems: The Cylc Workflow Engine. *Computing in Science Engineering* (2019), 1–1. https://doi.org/10.1109/MCSE.2019.2906593 00000.
29. Perkel, J.M. Workflow systems turn raw data into scientific knowledge. *Nature* 573 (September 2019), 149–150. https://doi.org/10.1038/d41586-019-02619-z.
30. POSIX.1-2008: IEEE Std 1003.1™-2008 and The Open Group Technical Standard Base Specifications, Issue 7. IEEE and The Open Group, https://pubs.opengroup.org/onlinepubs/9699919799.2008edition/.
31. Seemann, T. Ten recommendations for creating usable bioinformatics command line software. *GigaScience* 2, 2047-217X-2-15 (December 2013). https://doi.org/10.1186/2047-217X-2-15.
32. Simonis, I. OGC Earth observation applications pilot: Summary engineering report. Open Geospatial Consortium Public Engineering Report OGC 20-073 (2020), https://docs.ogc.org/per/20-073.html.
33. Taylor, R.C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics 11*, 12 (December 2010), S1. https://doi.org/10.1186/1471-2105-11-S12-S1.
34. van Wezenbeek, W.J.S.M., Touwen, H.J.J., Versteeg, A.M.C., and van Wesenbeeck, A.J.M. National Open Science Plan. Ministry of Education, Culture, and Science, Netherlands, (2017). https://doi.org/10.4233/uuid:9e9fa82e-06c1-4d0d-9e20-5620259a6c65.
35. Vivian, J. Toil enables reproducible, open source, big biomedical data analyses. *Nature Biotechnology 35*, 4 (April 2017), 314–316. https://doi.org/10.1038/nbt.3772.

more online

For additional information, access the supplementary material for this article at https://dl.acm.org/doi/10.1145/3486897.

Michael R. Crusoe is a promovendus at VU Amsterdam, Department of Computer Science, Netherlands.; CWL Project Lead at Software Freedom Conservancy, Inc., USA; and Project Leader Compute Platform in ELIXIR-NL at DTL Projects, Utrecht, Netherlands.

Sanne Abeln is an associate professor in Bioinformatics at VU Amsterdam, Department of Computer Science, Netherlands.

Alexandru Iosup is a university research chair and full professor at VU Amsterdam, Department of Computer Science, Netherlands.

Peter Amstutz is a principal software engineer at Curii Corporation, Sommerville, MA, USA.

John Chilton is a computational scientist at the Nekrutenko Lab at Pennsylvania State University Department of Biochemistry and Molecular Biology, State College, PA, USA.

Nebojša Tijanić was a software engineer at Seven Bridges Genomics Inc., Charlestown, MA, USA.

Hervé Ménager is a research engineer at the Institut Pasteur, Université de Paris, Bioinformatics and Biostatistics Hub, F-75015, Paris, France.

Stian Soiland-Reyes is a technical architect at The University of Manchester, Department of Computer Science, Manchester, U.K.; and a Ph.D. candidate at the Informatics Institute, University of Amsterdam, Netherlands.

Bogdan Gavrilović is a product development director at Seven Bridges Genomics Inc., Charlestown, MA, USA.

Carole Goble (CBE FREng FBCS CITP) is a full professor of Computer Science at The University of Manchester, Department of Computer Science, Manchester, U.K.