Reusability First: Toward FAIR Workflows

Matthew Wolf* 0000-0002-8393-4436
Jeremy Logan* 0000-0003-1529-3048
Kshitij Mehta* 0000-0002-9714-9981
Daniel Jacobson*‡ 0000-0002-9822-8251
Mikaela Cashman* 0000-0003-0620-7830
Angelica M. Walker‡ 0000-0003-4308-6302
Greg Eisenhauer† 0000-0002-2070-043X
Patrick Widener§ 0000-0002-5882-0816
Ashley Cliff‡ 0000-0001-7809-5546

*Oak Ridge National Laboratory
Oak Ridge, TN

†Georgia Institute of Technology
Atlanta, GA

‡Bredesen Center for Interdisciplinary Research and Graduate Education
University of Tennessee, Knoxville
Knoxville, TN

§Sandia National Laboratories
Albuquerque, NM

***Abstract*—The FAIR principles of open science (Findable, Accessible, Interoperable, and Reusable) have had transformative effects on modern large-scale computational science. In particular, they have encouraged more open access to and use of data, an important consideration as collaboration among teams of researchers accelerates and the use of workflows by those teams to solve problems increases. How best to apply the FAIR principles to workflows themselves, and software more generally, is not yet well understood. We argue that the software engineering concept of technical debt management provides a useful guide for application of those principles to workflows, and in particular that it implies reusability should be considered as 'first among equals'. Moreover, our approach recognizes a continuum of reusability where we can make explicit and selectable the trade-offs required in workflows for both their users and developers.**

To this end, we propose a new abstraction approach for reusable workflows, with demonstrations for both synthetic workloads and real-world computational biology workflows. Through application of novel systems and tools that are based on this abstraction, these experimental workflows are refactored to right-size the granularity of workflow components to efficiently fill the gap between end-user simplicity and general customizability. Our work makes it easier to selectively reason about and automate the connections between trade-offs across user and developer concerns when exposing degrees of freedom for reuse. Additionally, by exposing fine-grained reusability abstractions we enable performance optimizations, as we demonstrate on both institutional-scale and leadership-class HPC resources.

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Scientific software, the data that it consumes and produces, and the insights that they together enable are at the core of many areas of modern science. Software development practices in computational science have grown organically from a variety of disciplinary, group-level, and even per-researcher standards and assumptions. However, this has led to the accumulation of significant and frequently overwhelming difficulties not only in maintaining individual solutions, but also in enabling collaborations within and across disciplines. There is a need for a new approach to the construction, deployment, and evolution of scientific workflows, software, and their interdependent data sets which addresses both the convergence of multiple scientific disciplines and the increasing prevalence of heterogeneous and extreme-scale computing technologies.

Workflows are a constant presence in everyone's day as the 'flow of work' that one undertakes, and yet they are difficult to clearly define and work with in a computational environment [1]. The variety of runtime engines [2]–[5], interfaces [6], [7], and efforts toward generality [8], [9] makes it exceedingly difficult to create a workflow instance in a particular system and then be able to share it with someone else or to reuse it after months or years have passed. However, there is a great need to increase the reusability not only of scientific data sets but also of the codes and environments that are built to create and consume that data. Building a better abstraction for creating, maintaining, and accessing Reusable Workflows is a key challenge for computational and computer scientists alike.

The practical root of the problem is that much scientific software begins its life in idiosyncratic, small-group investigations that leave the resulting code as a black box for most outside researchers or developers. Research software engineering [10] (RSE) teaches practices for constructing software from reusable, testable, and shareable components, and this enables an ecosystem of tools that can enforce standards and practice. Naively implementing this would require a complete rewrite and reworking of much of the practical code being used to do science today, and the RSE practices generally focus on software components rather than whole workflows. Thus we are focusing on a solution that enables low-to-zero-cost

Funding was provided by the Plant-Microbe Interfaces (PMI) SFA, the Exascale & Petascale Networks for KBase project and by The Center for Bioenergy Innovation (CBI). These are all supported by the Genomic Sciences Program of Office of Biological and Environmental Research in the DOE Office of Science. This work was also supported in part by the joint U.S. Department of Veterans Affairs, US Department of Energy MVP CHAMPION program, and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

entry for software where code can easily begin in a black-box configuration and progressively expand to become more 'white-box', reusable, and performant.

The publication of the FAIR principles [11] helped crystallize a broad coalition of scientists, policy makers, and innovators around specific ways to make open science data more Findable, Accessible, Interoperable, and Reusable. There has been a great deal of conversation particularly on the last two points of interoperability and reusability for data sets, but there are at least emerging standards for how to achieve those for data. Attempts to broaden these principles of open science for research software [12] and workflows have had more limited success, particularly because the concepts of interoperability and reusability are significantly more complex when one is talking about actions that one can take, rather than tracking the behavior of data objects upon which those actions take place. That is the crux of the slow emergence of a community consensus on what reusability should mean for research software, and the situation becomes even more complex when trying to reason about reusable workflows [13].

From a practitioner's point of view, this issue is well known. Technical debt [14], [15] is a common concept that can be formulated as tracking the degree of human effort needed to repurpose or reuse a piece of data or code. Anything that isn't explicitly implemented in the item incurs technical debt. The need to run down the hall to ask how to find an experiment in the old file-naming scheme, restructuring a build system so that it will work on a new machine, or the fragile library dependency on one particular old version of a math library all require human time to service the technical debt incurred by the previous approach. In all of these exemplars, the key commonality is in the human intervention. Technical debt must be serviced when the data, software, or workflow is reused (by others or simply in a new context), and we contend that meeting the goals of a FAIR workflow in particular go beyond insuring efficient human intervention for reuse to structuring metadata catalogs to offer new abstractions for automation. If the mechanics of reusability can be partially or fully automated through new abstraction approaches, both at initial creation and at the time of reuse, the barriers for driving community interactions and innovations are lowered as a result.

As desirable as reusability is as a concept, it is exceedingly difficult to define exactly what might represent reusability in practice. For one scientist, it might mean that there is an easy way to transfer someone else's successful Parsl [3] workflow into something that Pegasus [4] would support. For another scientist, the goal of reuse would be to encapsulate both the imported and pre-existing workflows so that the Parsl workflow fragment would remain as is and just exchange data cleanly with the pre-existing parts of their workflow, regardless of runtime or specification language. In many cases, reusability does not require any change in software environment at all but instead is part of the normal exchange within a research collaboration as people join or leave. Our view is that none of these types of reusability is the correct one to use as a basis for measure (or indeed any of the many other possible variants). Instead, the key insight is that reuse represents a continuum of actions that may require human intervention or may be automatable, and no single metric should be expected to assess whether a particular factorization of a workflow is ideally reusable. Instead, we propose six gauge properties, outlined in Box I, for understanding reusability of workflows. Each of these properties represents a category axis along which one can track the progress of a workflow toward reusability.

Box I: Gauge Properties for Reusable Workflows.

- Data:
  - Access
  - Schema
  - Semantics
- Software:
  - Granularity
  - Customizability
  - Provenance

The six properties are aimed at representing actionable metadata characteristics that can be attached to data and computational aspects of workflow components, respectively. For example, we define properties for data access, data schema/format structure, and data semantics. Each collection of properties has multiple stages; at a base level, perhaps nothing is known about any of them. The next step up on data access might be to understand the basic protocol (e.g., POSIX file, zeroMQ queue). From there, increasing levels of the gauge would reflect further knowledge of the data I/O interface (e.g., CSV file, HDF5). Similarly, the other data aspects of schema representation and data semantics have multiple stages of increasing explicitness of representation, and the software aspects can be tracked in how they represent the granularity, customizability, and provenance of the execution environment. Further details on these six gauge properties and how they might be applied to system abstractions are presented in Section III.

Although reusability's intersection with technical debt should be familiar to any computer or computational scientist, we outline several specific examples that we can leverage to illustrate the technical contributions of our workflow abstraction in Section II. For each of these, we highlight not only the challenges of these exemplar science workflows but also the opportunities for automation in enhancing reuse within their respective communities. These examples serve as a reference point as we then further explore the implications of the abstraction and its six gauges in Section III. From there we describe the tools and interfaces that we developed and/or leveraged for this work in Section IV before turning to experimental evaluations in Section V. These experiments demonstrate that introducing higher-level models for the creation and composition of workflow components can improve reusability based on our multi-axis system without harming overall performance; indeed we show in some cases that the resulting workflows are more performant.

II. Motivating Scenarios

Our highlighted science use cases and scenarios all use workflows that combine aspects of human-in-the-loop and

automation in their components. Although each of these can (and have) been the basis of papers in their own right, here we focus specifically on the community-driven scenarios and their associated issues surrounding the technical debt/automatable reuse trade-off, establishing a clear basis for further discussion.

A. Genome-Wide Association Studies at Scale

Genome-Wide Associate Studies (GWAS) are a standard computational biology technique to identify genotype to phenotype associations. A typical use of GWAS is to use mixed linear models to associate single nucleotide polymorphisms (SNPs) to a phenotypic trait such as a disease state or other properties of an organism, to help discover the genomic architectures that contribute to phenotypes. Software tools used for GWAS analysis require specific formatting of the input data. Raw genotype and phenotype data need to be reformatted by the user for tool use. This type of data formatting task is not limited to GWAS; data wrangling is usually a time-consuming process in data science workflows, often taking up to 80% of the time in any given project. This phenomena, often referred to as the 80/20 rule of data science has been highlighted by the National Academies of Sciences, Engineering, and Medicine [16] and thoroughly discussed in the data science pedagogical literature [17]–[22].

Data formatting tasks in bioinformatics can take on several challenges. There can exist multiple formats for single types of data (e.g. genome annotations can be in BED, GTF2, GFF3, or PSL formats), data formats may not be consistent between workflow steps, and custom format structures can be used for specific use cases (e.g. to accommodate sparse data, different computational systems, or incorporate unique data fields). Handling this lack of consistency is often left to the bioinformatician. In cases where automated conversion tools do not exist, the researcher may create their own. However, this can come at a time and monetary cost, and often custom tools are poorly tested [23] which could result in downstream consequences such as incorrect scientific conclusions. Furthermore, custom tools may be created to only fit certain types and sizes of data, as well as to adhere to specific computing systems. In many cases, modular design of custom tools falls victim to the available resources of the researcher. Data processing and workflow software are also often shared both within and between research groups. For example, it is increasingly common that publications require releasing related source code. This further highlights the need for flexible software that is reusable across users and systems. For these reasons, bioinformatics users could benefit from frameworks that encourage automation and customizability.

GWAS provides a good example of the need for reusability through automation in scientific workflows. Here, scripted, multi-step workflows are often reused to process different data sets. Much of the workflow is static from use to use, but new data sets must be preprocessed to work with the static workflow components. These preprocessing workflows are typically under-engineered, with exactly enough effort applied to process the current dataset, creating technical debt with corresponding costs that must be incurred during any subsequent use.

B. iRF-LOOP for Genome Knowledge Graph Generation

Iterative Random Forest Leave One Out Prediction (iRF-LOOP) is a form of machine learning algorithm that can extract explainable properties of the datasets. It can be used on a multitude of data sets to produce all-to-all associations [24], [25]. Using a matrix with n features and m samples, iRF-LOOP will treat each individual feature as the dependent variable, or Y vector, and create an iRF model with the remaining n-1 features as the independent variables, or the X matrix. Like iRF, iRF-LOOP can produce meaningful insights even in cases where n is much larger than m. The result of each individual iRF run is a vector of size n of the importance for each independent feature in predicting the Y vector. Following the completion of the n individual iRF runs, the n importance vectors are normalized and concatenated into an n x n directional adjacency matrix, with values that can be viewed as edge weights between the features.

In practice, the iRF-LOOP model is created by running a separate iRF process for each dependent variable. The larger the set of features, the more iRF runs that are needed. Due to the different requirements of different HPC systems, setting up the runs themselves involves manually assigning the runtime parameters and gauging the compute resource division, and then manually creating the submit scripts for all of the iRF runs. Because the run times between the individual iRF processes can differ within one submission, estimating run times for a set of features becomes difficult. Due to the static nature of the submit script submission process, the run time differences can lead to idle nodes. Once a submission has completed, a list of failed runs is manually curated and requires a new submit script to be created and resubmitted.

Because of these manual requirements, the user must have a deep understanding of the process to appropriately create the model. Teaching new users itself adds time to the process and may result in incorrect parameters or run set ups if rushed, leading to more allocation usage and more wasted time. A model-driven, reusable workflow framework for building the iRF-LOOP model would better enable sharing the program with others and speed the reuse of the model even by the original team when applying it to new data or new hardware.

C. Model-driven Abstractions for Codesign Campaigns

Codesign evaluations differ from regular simulation runs in that they are meant to explore a wide parameter space spanning the application, middleware, and system layers, as opposed to conducting a few runs using the main simulation. The output of a codesign campaign is a catalog that describes the impact of different parameters on different output metrics. A codesign campaign tool is thus different from several existing tools in the scientific community [26] that are mostly focused on one-off workflow executions or simple ensemble-based runs. A codesign abstraction that allows declaring an *objective* of the study using different metrics such as searching for optimal

runtime, minimizing storage space, reducing communication overhead etc. can further help build high-level composition and query interfaces with more portable and reusable workflows.

Building a new workflow tool for codesign studies from the ground-up is both cumbersome and expensive. Several components for running codesign workflows already exist across the wide spectrum of currently available workflow tools. It is thus of interest to adapt a model-driven approach towards designing workflow technologies for codesign campaigns to be able to reuse existing software. Reusability being a primary metric, defining clear interfaces and models for using existing tools forms our core motivation.

We build upon our experience developing the Cheetah and Savanna suite [27] of workflow tools for conducting codesign studies for online data analysis and reduction for large data as part of the Exascale Computing Project. Cheetah and Savanna are designed around the fundamental 'Campaign' model, which allows end users to compose and execute campaigns of experiments that evaluate the performance of various parameters. Parameters of interest are scattered across the application domain (coupling and other ways to tune application behavior), middleware (data streaming, staging, file I/O), and the underlying distributed system (concurrency, process placement). Abstractions that facilitate reusability can be grouped in the composition and execution layers of the toolset. Because multi-disciplinary exchanges are so important to codesigning for exascale computing, reuse and interoperability have been central system requirements throughout.

III. Model

One of our key goals is to tease apart unnecessary couplings between the lifetime, visibility, and modality of software artifacts. Technical debt results in part from all artifacts being expressed in the same modality (a 3G language like C++ or Python), even if they are produced from different levels of abstraction (hand-tooled by a developer vs generated code). A consequence of this is that reusability is considered only in the contexts supported by those lower-level expressions, such as linking against third-party C++ libraries or using `pypi` or `pip` to add packages. Considerable software scaffolding efforts at various levels attempt to compensate: popular C++ class libraries like Trilinos, package managers such as Spack, software configuration frameworks based on CMake, HPC-compatible recontextualizations of software reuse constructs such as containers. These efforts add to the corpus of code that humans must maintain, but are also ideal participants in an automated workflow generation ecosystem. Automation provides an opportunity to directly address their contributions to technical debt – no debt accrues from code that can be efficiently deleted and regenerated when needed. In sum, we want to differentially raise both the level of abstraction in which a particular software expression becomes permanently stored and reused, and the level of automation supporting its development and deployment.

Here, we achieve the goal of raising the abstraction by laying out specific stages within each of our gauge categories. Our basis for this model comes from a wealth of experience with not only the DAG and Bag-of-Tasks workflow engines but also from our investment in streaming, in situ, and online workflows for Big Data, both of the scientific and the cloud-scale kinds. Although the specific levels of each gauge may seem somewhat abstract, each of them has been chosen so that they can represent specific, testable, and modelable metadata for the constituent workflow, executable, and data quantities.

This approach does not require a user to adopt a 'one right' platform or approach. Instead, a progressive characterization of the functional workflow models produces a reusability context for whatever code and data is generated. Further, we demonstrate how these gauges and models can aid in the reusability and portability through a particular set of tools, but the metadata expressions are intended to be open for any tools or users within the broader ecosystem.

For each of the gauges as listed in Figure 1 we have included some of the lowest tier levels of the assessment that we have identified. These are not intended to be exhaustive lists; rather, as models for autonomous reusability and interoperability become more sophisticated, we expect that there will be continued extensions and refinements on the gauge properties. We discuss each of them in greater depth here.

Data Access: As described in the introduction, the data access gauge is useful for assessing the degree of explicitness and automatability of the access to data. The type of representation (e.g., POSIX file or database), the library interface(s) available to interface it (e.g., POSIX read, HDF5, ADIOS, or mySQL), and the types of data query (e.g., linear access, random element access, or SQL query) are all necessary information if one were to automatically construct new interfaces to reuse pre-existing work. As you reach higher tiers of the gauge, there are dependencies on other gauge properties. For example, to capture information on a relevant SQL query beyond just its existence, one would need some minimal degree of data schema characterization to be available.

Data Schema: Schema management is probably the most familiar of the six gauges. Workflow components produce and consume data in a variety of formats, ranging from simple strings of bytes to more complex structures such as typed arrays, tables, graphs, meshes, and many others. Such data objects can be represented with file formats ranging from human-readable ASCII formats (e.g. CSV, JSON), to self-describing binary formats (ADIOS, HDF5), to custom binary formats (e.g., MatML). The more sophisticated the schema information, the more full-functioning other automated services can be in creating automated format conversion, templatized configurations, and other similar requests that are supported through other gauges.

Data Semantics: In our abstraction, the data semantics gauge is used to assess and capture the semantics of intended use or production of data, independent of any specific software that is consuming it. Is ordering important? Are data items consumed in a window or element by element? There is an entire ontology of automatable format transactions that can be explored in future work, but here we lump all of these together

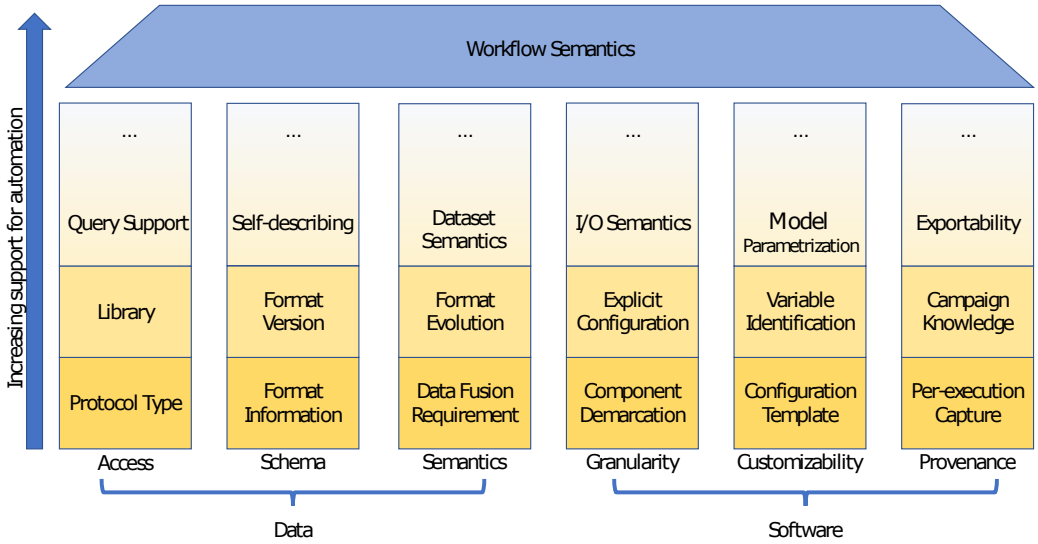

Fig. 1. Example properties for assessing workflow automatability using the six gauge principles.

into the 'data fusion' category. At a next layer above, the 'format evolution' tier leverages format version information to capture the conversions that would take a particular materials format back to an earlier version. Finally, information on how individual data elements should be engineered to be part of a complete dataset are in the data semantics tier. For example, designating a set of images as containing cancerous and healthy tissue samples (useful for a training workflow on segmentation algorithms) would be expressly captured through the dataset semantics gauge.

Software Granularity: The granularity of a reusable workflow component may be a code fragment, an individual executable code, a bundled workflow, or an internal service. This flexibility allows a workflow to evolve its reusability expression over time; you can always start by describing the entire multi-tier operation as a single component before progressing to teasing it apart. At whatever scale the constituent components are captured, making the configuration support explicit allows for the creation of templates for building, launching, and execution of the component. In order to support more automatable reuse, however, it is useful to capture the I/O semantics of the component, which needs to leverage rich information about the schema and semantics of the data it is consuming. For example, the component may process data one element at a time, but the first data element read is used for calculating deltas against all subsequent data. This 'first precious' data semantic would impact correctness if it is not respected in reuse scenarios, but it is also easy enough to capture as part of a machine-actionable plan for deployment.

Software Customizability: Customization of a reusable software component requires packaging that makes explicit which configuration characteristics can be modified as part of packaging the component for others to use. Within that larger list, there is a subset of relevant variables that reflect how a component might need to be customized. As we will see in the following section, we leverage previous work with the Skel system for model-driven code generation as an example for how such variable identification can be formalized into a machine-actionable model. At the next tier of model parameterization, the customization profile would also include understanding of how different variables are related to one another, and to how they change in the context of the campaign setting as laid out in the Provenance gauge's Campaign Knowledge tier.

Software Provenance: Although we place Provenance under the software banner, it is perhaps better to leave it unlabeled as just "Provenance", since it touches all of the areas above. The literature on provenance is deep, and we will only do a surface summary in this paper. What is most relevant for our model is to recognize that there are some key differences to reasoning about automation for reusability and interoperability of provenance data. One needs the standard provenance data and logs for each component and execution instance, but to support better automation, it is helpful to also have explicit context for the campaign in which that execution took place [28]. Tools can then summarize, evaluate and enable queries over hetereogeneous provenance logs. In order to go beyond self-usage, it is also important to capture what we have labeled as "Exportability". Intuitively, not all provenance that is useful to the original author is appropriate to include in a distributable, reusable research object. However, some provenance is crucial when reusing workflow components in a new context. So the policies of tracking the amenability and relevance of the gathered provenance for the generation of reusable components is tracked through this exportability tier.

A. Model Implications

There are two important facets to this approach. First, we are using the term gauge rather than metric because we feel it is important to distinguish between a measure that allows one to track the provenance of change toward community embrace of a particular workflow and a measure that would

provide an absolute score of 'reuse' to compare any two arbitrary workflows. Although this may appear as a needless distinction, it helps to address the key difficulty in discussing the human-in-the-loop aspects of technical debt management and reuse. There isn't an ideal amount of automation that would apply uniformly across all scientific software systems; some problems will always require more investment of human insight. A metric capable of comparing a biological machine learning workflow and an astrophysical image processing task might reasonably be created, but we assert that it would be less useful than something more descriptive and actionable.

Second, the particular gauge properties we've identified are each tied to abstract extensions that can aid in the creation of new runtime system optimizations. For instance, not only is data access a gauge property that can demarcate changes in how explicit the representation of access (e.g., it's a POSIX file, but no info on whether it's a custom binary blob or some fixed format). It also defines an ontology of terms that can be mapped into machine-queriable form. The gauges are useful from a human-driven provenance auditing perspective, while they can also be made machine-actionable. From this and the tools described in the following section, we construct a framework resembling a model-driven software engineering approach. It represents domain-specific computational elements as first-class, reusable entities from which specific executable expressions can be generated. Using such a framework, we can realize the benefits of model-based design in the definition of science workflows or ensemble computations.

IV. Tools & Environments

To provide some concrete implementation of the concepts discussed above, we have leveraged a collection of existing tools offering features that support the FAIR principles. In addition to the tools we have developed that are shown in more detail below, we have also exploited best practices for using standard tools and environments within the templates, models, and workflow components that we demonstrate in Section V. We leverage a number of standard features and libraries of scientific python usage, including Dask, numpy, and pip, but in a context where we can use the enhanced metadata of the reusability gauge abstraction to manage the technical debt associated with their use. Similarly, we exploit some of the advanced data access and schema markup capabilities of the Adaptable I/O System (ADIOS) while managing the configurational challenges of deploying it in novel circumstances. We have deployed customized versions of several tools as well to achieve our goals, as follows:

Skel: Model-driven Code Generation: Skel is a generative tool originally developed to produce skeletal I/O applications [29], [30]. Skel provides a model-driven code generation mechanism that couples a model of a desired action with one or more textual templates that drive the creation of files that implement the action. By defining a model that is a concise representation of the user decisions required for an action, and automating the way that the elements of the model impact the code, we can avoid the need for a user to have extensive interactions with the code itself. The user simply updates the model to reflect the current task, and the implementation is regenerated to reflect the new task.

Cheetah: Workflow Campaign Composition: The composition aspect of the toolset allows the science user to construct codesign campaigns aimed towards exploring several parameters. To build a high-level model, Cheetah's [27] composition interface provides an API that allows focusing on expressing parameters across the software stack, while omitting low-level system details from this high-level interface. The composition engine further adopts its own directory schema to represent a campaign end-point. The directory hierarchy represents simulation runs, and campaign metadata is hidden from the user. An API to submit a campaign and query its status is provided to investigate and interact with the campaign. It is important to note that the end user is focused on the description of the codesign process as opposed to fine details about instantiating a campaign, its storage schema, and its execution.

Savanna: Campaign Execution: Savanna [27], the execution engine of the toolset, runs all experiments in a campaign on the target system. It translates a high-level campaign description into actual system and scheduler calls, and provides a simple pilot runner to run experiments on available resources. Cheetah and Savanna communicate via an interoperability layer designed to represent an abstract manifest of the campaign. This layer implements a JSON schema to describe the full campaign, which includes the science applications, parameter sweeps declared by the user. While Savanna provides a simple job runner for the campaign, this design allows us to import existing workflow tools that provide efficient implementations for workflow patterns such as bag-of-tasks, pilot-based system, large-scale MPI runs etc.

V. Experiments

Each of our experiments has been chosen to demonstrate some of the practical implications and systems capabilities that arise from using the reusability gauge abstraction. The first two experiments are constructed to highlight specific capabilities of individual gauges, while the last two synthesize complex capabilities and solutions for services and real-world applications, respectively.

A. GWAS: Improving reusability at input

As a first demonstration, we built upon an initial implementation of GWAS workflow components as described in Section II-A. The goal was to increase its software granularity and customizability gauge values to enhance reusability. In the workflow we examined, one particular step involves *column-wise* pasting of a large number of individual tabular files into a single large file. We used the UNIX paste command to accomplish this column-wise pasting, but the size of the datasets generally precludes using a single paste. In fact, the paste operations become very slow if too many files are merged at once. Thus there was a two-phase paste, where a series of "sub-pastes" were performed to reduce the number of files, then a final paste was done to merge the pasted subsets.

In practice, this simple concatenation step is a fairly human-centric process, for several reasons. First, careful planning is required to divide the pasting into parallelizable subjobs that each have a reasonably short runtime. This is necessary not only for parallelization, but also to avoid filesystem bottlenecks from working with a large number of files simultaneously. Though setting up and running each of the subjobs that are needed to accomplish the preparation of a large data set requires only a small amount of thought and attention, this attention is spread over a longer period because successive queued jobs are run only after an indeterminate delay. Even preparing all jobs ahead of time, the scientist must check to see that jobs are completing successfully and keep track of which jobs remain to be submitted.

As such, the workflow becomes difficult to package for reuse, as so many of the steps involve unwritten, human-only operations. In order to eliminate this user overhead and opportunity for errors, we leveraged a generative solution to capture the correct granularity and customizability of the workflow components. Specifically, we used two tools, Skel and Cheetah, in concert.

We have defined a focused model for the paste operation that allows us to specify input and output data sets. The model includes information about the dataset under consideration (path and naming conventions), machine-specific details about resources to be used (such as limits on number of nodes and walltime, and relevant user account), and strategy for pasting. This model is provided as a JSON input file and is the single point of user interaction to specify the current problem specifications. Skel is then used to process the model by instantiating a set of templates to produce the set of files that implements a concrete workflow. These files include simple scripts that perform pasting of subsets and the final joining of the pasted subsets, a cheetah specification that details the set of individual tasks to be performed, as well as scripts for controlling and querying the progress of the work.

The model provides a focused point of interaction for the user, and reduces the need to parse and understand the structure of the code repeatedly during use. A side-by-side representation of the two formats is shown in Figure 2. In a traditional script, the user must fix any job scheduler parameters, directory paths, and hard-code any partitions of the data. Once complete, the user runs a series of separate jobs, each of which requires several manual perturbations of the script. In our Skel-driven approach, the user only modifies the script once, creating a cheetah campaign that manages running all of the necessary tasks during one or more jobs.

B. Dynamic Checkpoint-Restarts in HPC Simulations

As an example of setting the right degree of granularity and customization for a workflow service component, consider the well-studied area of dynamic checkpoint-restart. Compared to solutions that treat it as a single library [31], [32], treating it as a workflow component with explicit, model-driven policies offers interesting reuse possibilities. This demonstration is, however, not intended to innovate in checkpoint-restart techniques, but instead to leverage the existing knowledge base to apply it in the context of workflow construction.

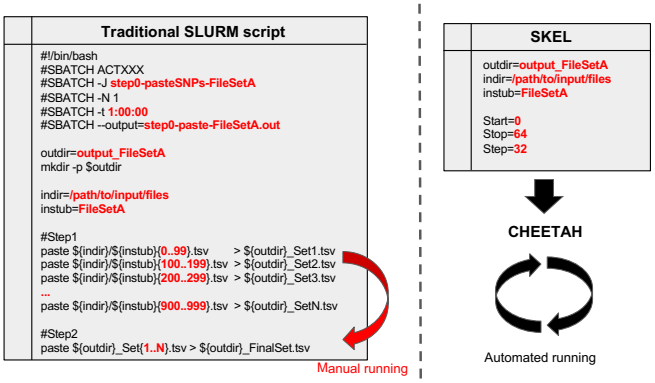

Fig. 2. A traditional manual script versus Skel-based automated script. Red text indicates fields or actions that require manual intervention by the user for a new run configuration.

A common practice is to implement a simple checkpointing mechanism in which a checkpoint is generated after a preset number of 'timesteps' in a simulation. This frequency is determined beforehand and depends on the failure rate of the underlying system and the overhead of checkpoint I/O. As checkpoint data is typically large in size, the output frequency is a trade-off between cost of I/O and cost of recovering from a failure. Consequently, the checkpointing frequency varies from one system to another, and requires tuning an efficient checkpointing frequency for a particular application use case.

It can be argued that this approach does not capture the true intent behind checkpoint-restarts. The number of timesteps between checkpoints is a representation of the wall clock time gap between checkpoints and the underlying characteristics of the system, such as the mean-time-to-failure (MTTF) and the transient load on the system. A dynamically adaptive checkpointing approach that automatically determines the right set of parameters for checkpointing based upon user input and tunes them at runtime based upon the state of the system can help build a reusable checkpointing system. While fully dynamic and autonomous checkpointing is non-trivial, exposing the right set of parameters can be a first step towards developing policies for checkpoint-restarts. We gauge the reusability of checkpointing policies using the manual effort required to determine the checkpointing mechanism used on the system for production runs.

Some of the application-level parameters that can be used to set policies are 1) wall clock time gap between checkpoints, and 2) acceptable overhead of checkpoint I/O. Policies can then be constructed using a combination of some of all of the exposed parameters. As an example, we present a use case where checkpoints are written using I/O overhead as the checkpointing condition. In our experiments, applications declare the maximum allowable checkpointing I/O overhead as a percentage of the total application runtime. The I/O middleware issues a checkpoint only as long as the current I/O overhead is within the preset value. Further fine-tuning may be done to ensure a certain minimum frequency of checkpointing.

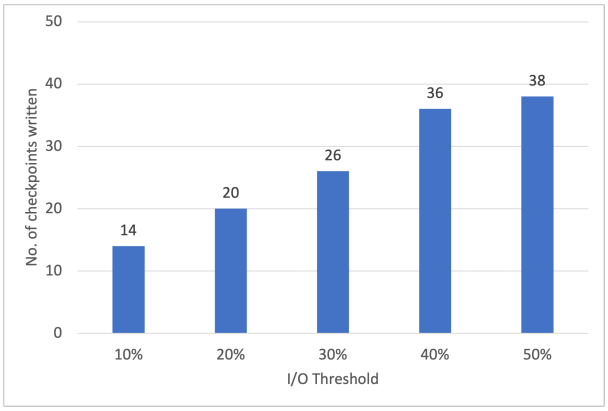

Fig. 3. Writing simulation checkpoints depending on the runtime overhead of checkpoint I/O, as opposed to writing a checkpoint every x timesteps. Exposing the I/O overhead as one of the parameters allows building flexible policies for checkpoint-restarts.

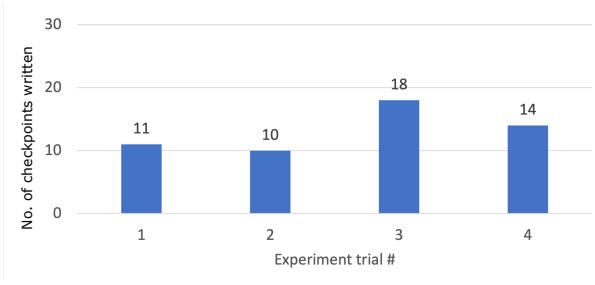

Fig. 4. The variation in the number of output checkpoints between multiple runs when maximum I/O overhead is set to 10% of the total application runtime. The overhead-driven policy takes into account runtime variations and changes in application behavior to determine checkpoint output.

The advantage of this approach is that the system may allow more frequent checkpointing if the cost of I/O is low, thereby allowing the simulation to restart from a more recent checkpoint in case of a failure. Additionally, the checkpointing library may set a policy in which an abnormally high I/O cost may be indicative of a system more prone to failure, and thus force a checkpoint to be issued.

Figure 3 shows the results of writing checkpoints based upon I/O overhead for a common reaction-diffusion benchmark on Summit, a leadership-class supercomputer at Oak Ridge National Laboratory. Depending on the runtime overhead, the I/O library determines if a checkpoint must be written to storage or not. Experiments were run using 4096 MPI processes spread evenly over 128 nodes. The application simulated 50 timesteps (thus, 50 maximum checkpoints possible), where each timestep generated a Terabyte of data. First, as expected, we observe that the number of checkpoints written to storage increases as the permitted I/O overhead increases. Figure 4 shows multiple runs of the application when the overhead is set to 10%. The changes in the number of checkpoints written is reflective of the changes in application behavior (configured to perform more/less computations and communication) and the state of the HPC system including the overhead on its file system.

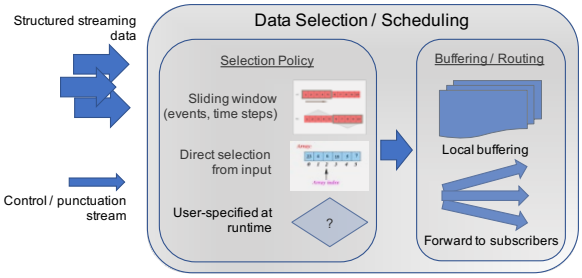

Fig. 5. Finer granularity in workflow construction allows greater reuse. In this instance, data selection criteria is separated from data movement infrastructure, reducing the amount of technical debt introduced by new workflow development.

C. Model-derived reusability in a synthetic workflow

The reusability gauge abstraction allows us to identify commonalities across workflows and encapsulate functionality which can be effectively reused. In a data-flow graph view of a workflow, such encapsulations appear as repeated subgraphs. Perhaps the most basic of these is a workflow in which data is collected in discrete units (e.g. individual observations or measurements) and forwarded to an aggregation or "data scheduling" component. Individual data items may be summarized, transformed, or left alone before being forwarded further along paths in the workflow graph. An important consideration in improving the reusability of a given workflow is the degree to which such structure and substructure (and the code that implements it) can be reused directly. In a collection/selection/forwarding workflow, the communication pieces (collection and forwarding) can be generated automatically given sufficient knowledge of data access patterns, data schema and semantics, as well as the degrees of granularity and customizability allowed by the software stack in use.

Consider the demonstration workflow of Figure 5 which represents data capture at an instrument and dissemination to one or more downstream consumers. In this workflow, all data formats are known beforehand, and so the communication code necessary can be automatically generated (given sufficient data description and marshalling support, complete a priori knowledge is not necessary even in high-performance binary data exchanges; cf. [33]). For this workflow to be maximally reusable, however, it has to be able to incorporate different selection policies at runtime – including policies not known at code generation or compile time.

In this example, the workflow begins with a simple data scheduling policy: forward each data item received to subscribers. A different instantiation of this data scheduling subgraph could be generated with a different policy which implements a sliding window based on time or item count over the incoming data. Using a skeleton-based approach for the communication components of the subgraph, while customizing the data scheduling action for each set of generated code, can be straightforwardly implemented in the Skel framework described earlier while providing efficient data transport and selection performance.

Further extending this example more completely illustrates the potential of combining model-driven code generation with run-time specialization of the resulting workflow. Consider a remote steering process which introduces a data scheduling policy into the workflow which was unknown at code-generation time. Using a control input (or "data punctuation" input, signaling abstract divisions between groups of data), a policy which implements (for instance) direct selection of queued data items is installed and made active. This is an extremely powerful mechanism on its own, enabling varied monitoring and steering inputs from outside the workflow which can themselves be informed by the data flowing through the graph. However, the combination of this capability with generated code provides exactly the benefits described above: reuse of code which does not change often (the communication components) and highly-granular specialization of code which needs to change at runtime (data scheduling).

This demonstration workflow supports the simultaneous installation of multiple data scheduling policies in its workflow subgraph; those policies can be selectively invoked using input from the control channel. In this way, the data scheduler implements a number of *virtual data queues*, each defined by its own selection policy. The usefulness of this approach is underscored by a history of developing efficient data-transfer middleware, from event-based publish-subscribe systems supporting rich subscriber customizations [34], to service layers providing flexible distributed rendezvous and notification facilities [35], to frameworks for structured data staging in high-performance I/O environments [36]. These systems offer the encapsulations and capabilities necessary to manage and implement virtual data queues and other structures, reducing the technical debt created by workflow development while remaining flexible enough to evolve with ongoing scientific inquiry.

D. Reusable Workflows for Iterative Random Forests for Predictive Expression Networks

As described in Section II-B, composing the iRF-LOOP workflow for predictive expression networks on a cluster requires a manual pre-processing phase in which the end user generates the execution scripts for the underlying system depending on the number of nodes allocated to the batch job. The script creates the directory hierarchy for the runs and submits them in groups or 'sets' with explicit synchronization at the end of a set. However, the explicit synchronization has the inherent disadvantage that all experiments in a set must be complete before the next set is run. Straggler processes can severely limit the performance of the overall workflow. Additionally, the researchers are responsible for tracking the progress of their jobs, and addressing failures and provenance for their large ensemble of runs. Furthermore, as the workflow is driven by constraints of the underlying system, it can vary from one system to another.

Creating a more portable and reusable workflow requires demarcation between a science application and the underlying system. We use the Cheetah and Savanna suite of tools that provide an abstraction to set up ensemble runs of parameters in an easy way while transparently handling job submission on the target system. We gauge the reusability of this system using the manual effort required to set up, track, and submit additional runs for different parameters using differently-sized allocations. This is determined by two aspects of the user's participation - 1) ease of setting up parametric runs, and 2) ease of submitting and tracking jobs on the system.

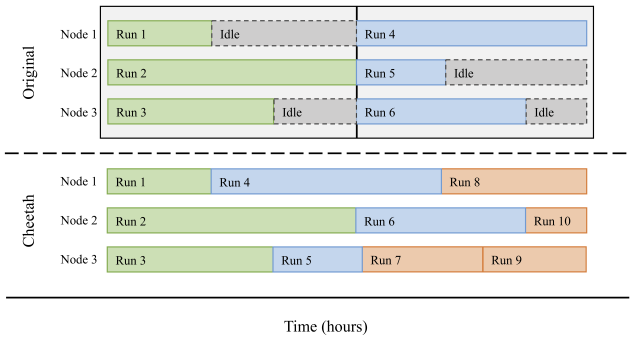

Fig. 6. Comparison of workflows between the original iRF-LOOP workflow and the improved Cheetah workflow. The original workflow required all runs within a set to complete before moving to the next set, resulting in idle nodes. This is eliminated using Cheetah.

We use Cheetah – the composition engine of the workflow suite – to create a campaign of iRF-LOOP runs. The Campaign abstraction in Cheetah allows creating a large ensemble study composed of one or more parameter 'Sweeps', which may be grouped into 'SweepGroups'. The Python-based composition API allows high-level expression of application-level, middleware-level, and system-level parameters. The Savanna workflow engine uses one of its several executor backends to translate the abstract campaign specification into low-level scheduler calls for the target system. It consists of a resource manager that dynamically schedules and tracks runs on the allocated nodes, thereby no longer requiring synchronizing runs and leading to better resource utilization. Users may simply re-submit a partially completed SweepGroup of parameters to continue execution. That is, the scientist's participation is limited to composing the parameter sweeps; execution is transparently and fully addressed by the workflow tool. Figure 6 shows how Savanna dynamically tracks the allocated nodes and leads to better overall resource utilization.

Figure 7 shows the performance gains obtained when using the Cheetah-Savanna workflow suite. It shows the time to run a Campaign on the 2019 American Community Survey produced by the United States Census Bureau [37]. Creating an all-to-all network of this data set can inform the user of relationships between the various demographic, socioeconomic, and housing features. 1606 features for 3220 counties were obtained using the R package `tidycensus` [38]. The Campaign specification creates a parameter sweep over all the 1606 features. Cheetah automatically generates directory hierarchy and metadata for the Campaign. Savanna launches the campaign and manages execution of individual runs. If all

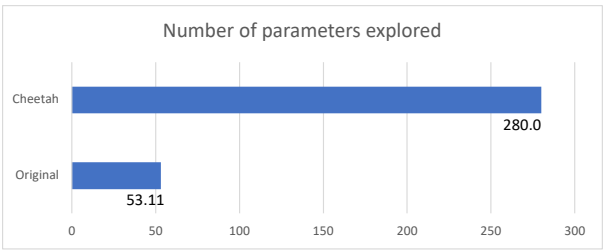

Fig. 7. Performance improvements in the iRF-LOOP workflow using the Cheetah-Savanna workflow suite. Values shown represent the average number of parameters explored in 2-hour allocations of 20 nodes on the Summit supercomputer at Oak Ridge National Laboratory.

runs in the SweepGroup cannot be run in the allotted time, the SweepGroup is simply re-submitted, and Savanna resumes execution of the experiments. We observe over $5\times$ improvement in total runtime using the Cheetah-Savanna toolsuite.

VI. Related Work

The literature on scientific workflows, open science, and effective systems for managing both of them is extensive. We have noted much of this prior art throughout this paper, and here highlight a handful of the relevant efforts that have helped inspire our approach. There are numerous projects aimed at workflow management, including Parsl [3], Pegasus [39], Swift-T [40], and Radical-Pilot [41] to name just a few. Maestro [42] is a workflow management tool with generative capabilities that also focuses on managing the execution of a collection of workflow components.

Many other efforts address efficiently managing data movement between processes and from process to disk, leveraging community standard data protocols and schemas. Related examples include Apache Parquet [43], a library for interacting with disk-based column stores; HDF5 [44], which provides a widely-used self-describing format organized around a hierarchical model; and the Adaptable I/O System (ADIOS) [36], a self-describing data solution offering storage and communication options for large-scale data. ADIOS provides a mature implementation of *data staging* [45], a technique for leveraging additional compute nodes to improve I/O performance on HPC platforms. Some existing codes might take advantage of this sort of library, although many still rely on custom I/O solutions or delimited text formats.

VII. Conclusion

Data-centric computing is increasingly important for scientific progress, and the FAIR principles have come to represent an important way forward for open science datasets. In this paper, we have shown that these same principles also need to be applied to the workflows that represent the interplay of consuming and generating those data sets. Leveraging the idea of technical debt from software engineering we recognize that the default for reuse always involves human intervention. Making software more reusable will therefore depend on making metadata explicit (so that it can be audited by humans), but also machine actionable (in order to significantly automate the management of technical debt inherent in the use of other's software and data). This is completely in keeping with the original definition of FAIR, and, as such, can be seen as a refinement of the requirements for community-specified metadata for Reusability and Interoperability (particularly points R1.2, R1.3, and I3 from [11]).

The six gauge principles – three for data (access, format, and semantics), three for software (granularity, customization, and provenance) – are specific components of metadata extensions that can be exercised by automated systems in order to better connect an original use to a reuse context. The examples we have described here demonstrate a variety of potential applications for this perspective. The GWAS workflow example shows the importance of data encapsulation as a vehicle for automating ingress of data to workflow components. With the co-design for checkpoint restart and the sliding window technical demonstration we show how system automation can be brought to bear in support of more extensive application of these explicit metadata characterizations. Finally, the iRF-LOOP workflow demonstrates the synthesis of applying both software and data encapsulation and how that synthesis not only makes the code more reusable but, in this case, also improves the portability and performance of the resulting workflow instance. Through application of novel systems and tools that are based on our reusability gauge abstraction, these experimental workflows are refactored to right-size the granularity of workflow components, efficiently filling the gap between end-user simplicity and general customizability.

Our work makes it easier to selectively reason about and automate the connections between trade-offs across user and developer concerns when exposing degrees of freedom for reuse. Looking toward future development, we see great potential for more powerful and granular metadata representation, automation of reusable workflow composition, and applications across diverse areas of computational science (including climate, materials research, computational systems biology, and hybrid experimental/simulation platforms). Workflows represent the connections between data, computation, and human decision-making, and making them more reusable and automatable will have benefits across the science ecosystem.

Acknowledgment

This manuscript has been authored by UT-Battelle, LLC under contract no. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan, last accessed September 16, 2020).

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525 (SAND2021-XXXXC).

REFERENCES

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future generation computer systems*, vol. 25, no. 5, pp. 528–540, 2009.

[2] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome biology*, vol. 11, no. 8, pp. 1–13, 2010.

[3] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, "Parsl: Pervasive parallel programming in python," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 25–36. [Online]. Available: https://doi.org/10.1145/3307681.3325400

[4] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[5] J. J. Billings, A. R. Bennett, J. Deyton, K. Gammeltoft, J. Graham, D. Gorin, H. Krishnan, M. Li, A. J. McCaskey, T. Patterson, R. Smith, G. R. Watson, and A. Wojtowicz, "The Eclipse Integrated Computational Environment," *arXiv:1704.01398 [cs]*, Mar. 2017, arXiv: 1704.01398. [Online]. Available: http://arxiv.org/abs/1704.01398

[6] J. Ison, M. Kalaš, I. Jonassen, D. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer, and P. Rice, "Edam: an ontology of bioinformatics operations, types of data and identifiers, topics and formats," *Bioinformatics*, vol. 29, no. 10, pp. 1325–1332, 2013.

[7] J. J. Billings and S. Jha, "Toward common components for open workflow systems," *arXiv preprint arXiv:1710.06774*, 2017.

[8] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic, "Common Workflow Language, v1.0." Figshare, 2016. [Online]. Available: https://doi.org/10.6084/m9.figshare.3115156.v2

[9] B. Chapman, J. Gentry, M. Lin, P. Magee, B. O'Connor, A. Prabhakaran, and G. Van der Auwera, "OpenWDL," 2019.

[10] C. R. Prause, R. Reiners, and S. Dencheva, "Empirical study of tool support in highly distributed research projects," in *2010 5th IEEE International Conference on Global Software Engineering*, 2010, pp. 23–32.

[11] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[12] A. lena Lamprecht, L. Garcia, M. Kuzak, C. Martinez, R. Arcila, E. Martin Del Pico, V. Dominguez Del Angel, S. Van De Sandt, J. Ison, P. Martinez, P. Mcquilton, A. Valencia, J. Harrow, F. Psomopoulos, J. Gelpi, N. Chue Hong, C. Goble, S. Capella-gutierrez, P. Groth, P. Groth, and M. Dumontier, "Towards fair principles for research software," *Data Science*, vol. 3, no. 1, pp. 37–59, Jun. 2020.

[13] C. Goble, S. Cohen-Boulakia, S. Soiland-Reyes, D. Garijo, Y. Gil, M. R. Crusoe, K. Peters, and D. Schober, "FAIR Computational Workflows," *Data Intelligence*, vol. 2, no. 1-2, pp. 108–121, 01 2020. [Online]. Available: https://doi.org/10.1162/dint_a_00033

[14] W. Cunningham, "The wycash portfolio management system," in *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum)*, ser. OOPSLA '92. New York, NY, USA: Association for Computing Machinery, 1992, p. 29–30. [Online]. Available: https://doi.org/10.1145/157709.157715

[15] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (dagstuhl seminar 16162)," *Dagstuhl Reports*, vol. 6, 01 2016.

[16] National Academies of Sciences, Engineering, and Medicine, *Data science for undergraduates: Opportunities and options*. National Academies Press, 2018.

[17] A. Y. Kim and J. Hardin, ""playing the whole game": A data collection and analysis exercise with google calendar," *Journal of Statistics and Data Science Education*, vol. 29, no. sup1, pp. S51–S60, 2021.

[18] B. Baumer, "A data science course for undergraduates: Thinking with data," *The American Statistician*, vol. 69, no. 4, pp. 334–342, 2015.

[19] J. Hardin, R. Hoerl, N. J. Horton, D. Nolan, B. Baumer, O. Hall-Holt, P. Murrell, R. Peng, P. Roback, D. Temple Lang *et al.*, "Data science in statistics curricula: Preparing students to "think with data"," *The American Statistician*, vol. 69, no. 4, pp. 343–353, 2015.

[20] A. Loy, S. Kuiper, and L. Chihara, "Supporting data science in the statistics curriculum," *Journal of Statistics Education*, vol. 27, no. 1, pp. 2–11, 2019.

[21] Y. Xie, "knitr: A general-purpose package for dynamic report generation in r," *R package version*, vol. 1, no. 7, 2013.

[22] N. J. Horton, B. S. Baumer, and H. Wickham, "Setting the stage for data science: integration of data management skills in introductory and second courses in statistics," *arXiv preprint arXiv:1502.00318*, 2015.

[23] D. Verma, J. Gesell, H. Siy, and M. Zand, "Lack of software engineering practices in the development of bioinformatics software," *ICCGI*, vol. 2013, pp. 57–62, 2013.

[24] A. Cliff, J. Romero, D. Kainer, A. Walker, A. Furches, and D. Jacobson, "A high-performance computing implementation of iterative random forest for the creation of predictive expression networks," *Genes*, vol. 10, no. 12, p. 996, 2019.

[25] S. Basu, K. Kumbier, J. B. Brown, and B. Yu, "Iterative random forests to discover predictive and stable high-order interactions," *Proceedings of the National Academy of Sciences*, vol. 115, no. 8, pp. 1943–1948, 2018.

[26] "Existing workflow systems," https://s.apache.org/existing-workflow-systems.

[27] K. Mehta, B. Allen, M. Wolf, J. Logan, E. Suchyta, J. Choi, K. Takahashi, I. Yakushin, T. Munson, I. Foster, and S. Klasky, "A codesign framework for online data analysis and reduction," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, 2019, pp. 11–20.

[28] J. Logan, K. Mehta, G. Heber, S. Klasky, T. Kurc, N. Podhorszki, P. Widener, and M. Wolf, "A vision for managing extreme-scale data hoards," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2019.

[29] J. Logan, S. Klasky, H. Abbasi, Q. Liu, G. Ostrouchov, M. Parashar, N. Podhorszki, Y. Tian, and M. Wolf, "Understanding I/O performance using I/O skeletal applications," in *Euro-Par 2012 Parallel Processing*. Springer Berlin/Heidelberg, 2012, pp. 77–88.

[30] J. Logan, J. Y. Choi, M. Wolf, G. Ostrouchov, L. Wan, N. Podhorszki, W. Godoy, E. Lohrmann, G. Eisenhauer, C. Wood, K. Huck, and S. Klasky, "Extending Skel to support the development and optimization of next generation I/O systems," in *2017 IEEE International Conference on Cluster Computing*, ser. CLUSTER '17, 2017, pp. 563–571.

[31] A. Moody, D. Sikich, N. Bass, M. J. Brim, C. Stanavige, H. Sim, J. Moore, T. Hutter, S. Boehm, K. Mohror *et al.*, "Unifyfs: A distributed burst buffer file system-0.1. 0," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2017.

[32] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello, "Veloc: Towards high performance adaptive asynchronous checkpointing at large scale," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 911–920.

[33] M. Wolf, Z. Cai, W. Huang, and K. Schwan, "Smartpointers: Personalized scientific data portals in your hand," in *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE, 2002, pp. 20–20.

[34] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: Opportunities and challenges at exascale," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '09. New York, NY, USA: Association for Computing Machinery, 2009. [Online]. Available: https://doi.org/10.1145/1619258.1619261

[35] F. Bustamante, P. Widener, and K. Schwan, "Scalable directory services using proactivity," in *SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, 2002, pp. 65–65.

[36] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, May 2014. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3125

[37] United States Census Bureau, *American Community Survey: Data Profiles*, 2019, Accessed: Mar. 11 2021. [Online]. Available: https://www.census.gov/acs/www/data/data-tables-and-tools/data-profiles/2019/

[38] K. Walker and M. Herman, *tidycensus: Load US Census Boundary and Attribute Data as 'tidyverse' and 'sf'-Ready Data Frames*, 2021, r package version 0.11.4. [Online]. Available: https://CRAN.R-project.org/package=tidycensus

[39] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, pp. 219–237, 2005.

[40] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/T: Scalable data flow programming for many-task applications," in *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2013, pp. 309–310.

[41] A. Merzky, M. Santcroos, M. Turilli, and S. Jha, "Radical-pilot: Scalable execution of heterogeneous and dynamic workloads on supercomputers," *CoRR, abs/1512.08194*, 2015.

[42] "Maestro workflow conductor: Developing sustainable computational workflows," https://computing.llnl.gov/projects/maestro-workflow-conductor, accessed: 2020-12-22.

[43] "Apache parquet," https://parquet.apache.org/, accessed: 2020-12-11.

[44] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the HDF5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, ser. AD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 36–47. [Online]. Available: https://doi.org/10.1145/1966895.1966900

[45] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky, "Just in time: Adding value to the IO pipelines of high performance applications with JITStaging," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, ser. HPDC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 27–36. [Online]. Available: https://doi.org/10.1145/1996130.1996137