Improving Publication and Reproducibility of Computational Experiments through Workflow Abstractions

Yolanda Gil
Information Sciences Institute
University of Southern California
USA
gil@isi.edu

Daniel Garijo
Information Sciences Institute
University of Southern California
USA
dgarijo@isi.edu

Margaret Knoblock
Information Sciences Institute
University of Southern California
USA
mrk022@bucknell.edu

Alyssa Deng
Information Sciences Institute
University of Southern California
USA
shipingd@andrew.cmu.edu

Ravali Adusumilli
School of Medicine
Stanford University
USA
ravali@stanford.edu

Varun Ratnakar
Information Sciences Institute
University of Southern California
USA
varunr@isi.edu

Parag Mallick
School of Medicine
Stanford University
USA
paragm@stanford.edu

ABSTRACT

The current practice of publishing articles solely containing textual descriptions of methods is error prone and incomplete. Even when a reproducible workflow or notebook is linked to an article, the text of the article is not well integrated with those computational components, and the workflow and notebook are focused mostly on implementation details that are disconnected from the scientific approach described in the text of the article. Through an analysis of three multi-omics articles, we illustrate why this makes it difficult to understand, reproduce, compare, and reuse computational methods. We propose workflow abstractions that that capture different concepts and perspectives that are important to scientists. These abstractions connect the text of an article to the corresponding workflow, and provide a framework to improve the publication and reproducibility of computational experiments.

CCS CONCEPTS

• **Information systems** → **Artificial intelligence**; *Knowledge representation and reasoning*

KEYWORDS

Reproducibility, semantic workflows, semantic science

2017 Workshop on Capturing Scientific Knowledge (SciKnow), held in conjunction with the ACM International Conference on Knowledge Capture (K-CAP), December 4, 2017, Austin, TX.

1 INTRODUCTION

The reproducibility crisis in science has received significant attention. Reproducibility requires that methods are described with enough details to repeat the experiment in an independent lab or setting. For computational experiments, studies show that published papers often provide insufficient information about the data, protocols, software, and overall method used to obtain the new results [Van Noorden 2015]. A major barrier to reproducibility can be traced to the traditional, unstructured format of publications "materials and methods" sections. The ambiguity, imprecision, and linearity of text make natural language descriptions of computational analyses inadequate for reproducible research [Steehouder et al 2000; Garijo et al 2013; Gil 2015; Groth and Gil 2009]. A major problem is that there is no guidance or methodology to describe computational methods in articles. It is unclear what the intent of the descriptions in methods sections is. Is the goal to provide a step-by-step account of the procedures taken, parameters employed, and data provenance such that a study might be reproduced? Alternately, is the goal to provide a high-level intuition for the steps that were performed? Both are valuable but are incompatible objectives in current text-based descriptions, leading to neither an intuitive reading experience nor a reproducible description.

Computational workflows and notebooks can be used to organize and record computational methods, and are often linked to publications. Workflows capture the dataflow among computations, so the different steps of the method are explicitly represented and linked. Notebooks are composed of cells that contain either text or code that can be easily re-run. Workflows and notebooks facilitate the documentation of the software and the structure of a computational method. But even when workflows and notebooks are used, the text for those publications is always manually generated by the authors and often inadequately captures the full complexity of an analysis, leading to poor reproducibility. In prior work, we developed an approach to automatically create descriptions of computational methods by generating text from workflows [Gil and Garijo 2017], where the text accurately represents what was done and can be presented from different perspectives. The text, however, can only be as good as the workflows that it was generated from. This motivates the need for a workflow design methodology that leads to workflow representations that support the automated generation of explanations as text that can be used in publications.

This paper proposes abstractions designed to improve the publication of computational methods to facilitate reproducibility. This methodology extends our prior work on representing and publishing workflow abstractions using community standards for workflows and provenance [Garijo et al 2017].

The paper begins with an overview of related work on reproducibility and publication of computational methods. We then present an analysis of the computational methods described in three seminal papers in cancer omics. We introduce the abstractions proposed, and discuss their merits in improving the descriptions of computational methods in scientific publications.

2 PUBLICATION AND REPRODUCIBILITY OF COMPUTATIONAL METHODS

Textual descriptions of methods in articles may be incomplete (e.g., [Ioannidis et al 2009; Donoho et al 2009]). Authors focus on conveying the major contributions of the work and describe the methods in that light, omitting details that may be important for transparency and reproducibility. For example, [Garijo et al 2013] describes our work to reproduce an article for which the authors had provided the data, software, and results to facilitate reproducibility. We created reproducibility maps, that showed different categories of users could figure out from the text of the paper how the work was done. The reproducibility maps showed that only researchers with the same level of expertise in the subject as the authors were able to figure out how to fully reproduce the work. There are many similar results in the literature, some mentioning the lack of publication of data [Ioannidis et al 2009] and others the lack of details in the description of methods leading to "exercises in 'forensic bioinformatics' where aspects of raw data and reported results are used to infer what methods must have been employed" [Baggerly and Coombes 2009]. There are several reasons why text descriptions of methods are riddled with problems. First, articles often have space limitations, so authors tend to omit anything that seems not important. Second, they are manually written without any particular guidance, it is easy for authors to provide imprecise descriptions. Finally, computational methods are often complex procedures with non-linear structures that are hard to describe with the sequential nature of text [Gil 2015]. Even when authors endeavor to describe enough details, textual descriptions are often ambiguous. A study reported in [Ince et al 2012] looked at writing software from scratch based on the textual descriptions reported in geophysics papers and found radical differences in the implementations. The papers were found to be ambiguous at the lexical, syntactic, and semantic level, and not necessarily because the authors were not rigorous but because natural language is inherently ambiguous. We also find that the methods sections of articles mix general methods with specific details of the executions carried out [Gil and Garijo 2017]. Although there are many tools and recommendations of best practices for authors [Stodden et al 2016], it is still up to them to figure out what to include in an article and its methods section. In summary, textual descriptions of methods in articles are far from ideal, since the text tends to be: 1) Incomplete, omitting important details about the computations performed; 2) Ambiguous, having several interpretations of how the computations were actually done; 3) Mingled, interspersing general overviews with execution details.

Workflows capture unambiguously a computational analysis as a dataflow among steps [Taylor et al 2006]. In prior work, we found that workflow reusability is a major drive for users [Garijo et al 2014a]. Workflow repositories provide mechanisms to publish and search workflows, particularly to improve reproducibility and sharing of computational experiments. However, the descriptions of workflows are manually generated and therefore are as incomplete as those in scientific articles. In prior work we

analyzed the textual descriptions of workflows from one of these repositories [Groth and Gil 2009]. We found significant differences between what was included in the textual descriptions and the actual formal specification of the workflows. A major limitation of workflow representations is that they mix major method steps with ancillary steps that do for example minor data reformatting. Also in previous work, we analyzed workflows to identify by hand general categories of steps (motifs) that make such distinctions [Garijo et al 2014b]. But workflows in themselves have no explicit mention of the relative importance of steps and all steps are treated equally. In summary, although workflows provide a formal computational representation of methods, the workflows themselves are: 1) Incomplete, because workflow representations do not express important semantic properties of steps; 2) Flat, with abstractions often absent from the workflow structure; and 3) Undifferentiated, as there is no explicit distinction between important steps and ancillary steps.

A recent popular trend is electronic notebooks, such as Jupyter Notebook and Apache Zeppelin, where the advantage is that the text is intermixed with data and code so it is easier to follow step by step how the method actually works. This approach is akin to executable papers which have been around for some time, such as Sweave and knitr which combine Latex and R [Xie 2015]. However, a reader cannot easily compare two notebooks, since that requires comparing the code line by line, and cannot easily reuse parts of one to create another since the code in notebook cells is not necessarily modular. In addition, although notebooks are easily published and shared they have not replaced published papers, possibly due to their idiosyncratic formats which do not yet offer the persistence and archival guarantees required by publishers.

In summary, in order to understand a published article, and assess its validity, reproduce the work, or to compare its method to another article, a reader must do a significant amount of work. Even when authors capture computational methods as workflows, there is no guidance on how to facilitate reproducibility and reuse. The next section analyzes specific articles in detail as motivating scenarios, and extracts desiderata for workflow design.

3 AN ANALYSIS OF MULTI-OMICS METHODS

This section motivates our work in the context of three seminal articles in multi-omics: [Zhang et al 2014], which is the first publication of a large-scale multi-omics analysis, [TCGA 2008] and [Imielinski et al 2012] which describe work on genomics that Zhang and colleagues built upon. A detailed analysis of all three articles is provided in [Knoblock 2017].

3.1 Method Descriptions

The methods section of a scientific article describes, together with the supplementary materials, the data and computational steps used for data analysis. We illustrate how methods are typically described using excerpts from [TCGA 2008] for variant calling from resequencing data, where SNPs and indels were screened against dbSNP for position/allele match. First, "*Putative variants were identified using Polyphred 6.1, Polyscan 3.0, SNPdetector 3, and SNP Compare. SNPs and indels were screened against dbSNP for position/allele match*". This excerpt describes the software, although it refers to entire packages and not how they are used in the method. Then, "*Boundaries of insertion, deletion and complex rearrangements [were] annotated*", and the detailed annotation guidelines are outlined in the article. That excerpt focuses on the science method. Next, "*The first step in analysis of the mutation data was to combine the .maf files from all centers into a .mut file containing at most one record for each site-sample pair. In the process of combining the files, care was taken to detect and resolve conflicts between multiple records for the same site-sample*". This excerpt is focused on low level implementation aspects such as file formats and handling duplicate entries. And finally, "*As part of our sequencing pipeline, non-synonymous mutations were subjected to an orthogonal validation or re-sequencing (verification) step to decrease the prevalence of false positives. In our analysis we considered only those mutations that were confirmed by validation or verification to be actual somatic mutations*". This excerpt focuses on the science aspects of the analysis, but it does not specify how each step of the method are implemented by the software packages mentioned earlier.

This is a common approach to describing methods. **Method descriptions in scientific articles mix mentions of software, data formats, and scientific descriptions of the experiment**.

Figure 1 shows a workflow that a biologist created based on the method description in the article. Not surprisingly, the workflow steps are also mixture of science concepts and software implementation and data formats. Note that this makes it hard for another scientist to understand and therefore reuse the workflow.

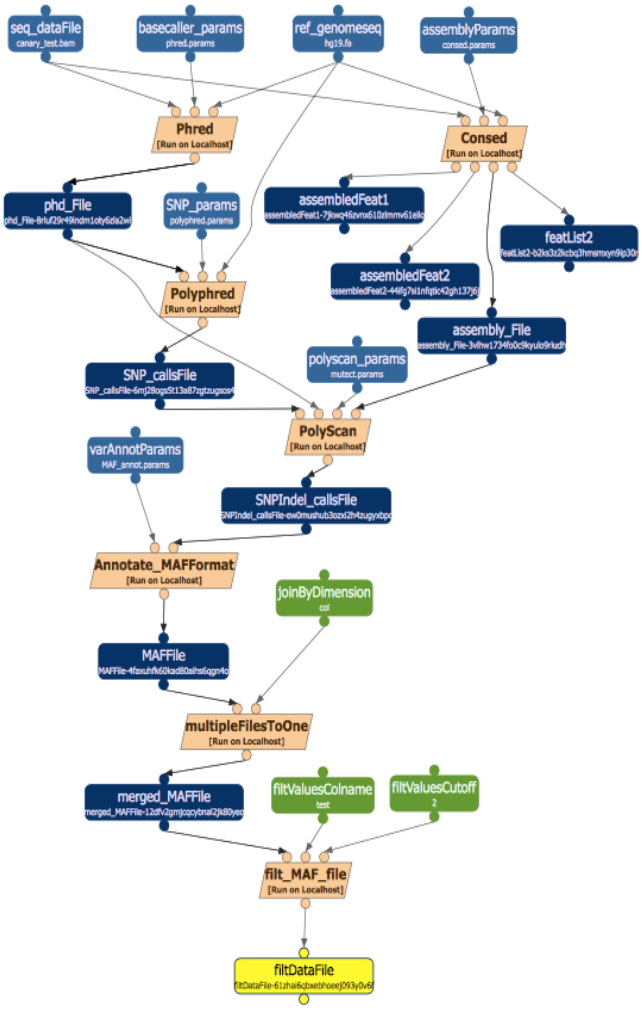

Figure 1: Computational workflow to annotate variant calling for resequencing data, based on [TCGA 2008]. Each workflow step (square boxes) is described as the software that implements the step.

A scientist may not be familiar with the different software packages (there are hundreds of packages that are available for this kind of analysis), and therefore would not understand the function of each step. Therefore, the design of workflows should accommodate the separation between the conceptual description of the experiment and the implementation of the experiment in software.

3.2 Software Descriptions

Each step in a method can be described at a conceptual level in terms of the function that it performs, and at an implementation level as the software used for the step. For the articles we analyzed, these descriptions are as follows:

- Base caller (Phred software): Assembling genome for genomic alignment/features.
- Genome assembly (Consed software): Calling genomic bases from input files.
- SNP caller (Polyphred software): Calling SNPs from the input genomic file (variant calling).
- Indel and SNP caller: (PolyScan software): Calling both indels and SNPs (variant calling).
- Variant annotation (Annotate_ MAFFormat software): Annotating variants based on reference genomes.
- Join data files (mutipleFilesToOne software): Appending multiple text files.
- Filter data files (filt_MAF_file software): Variant filtering based on input parameters.

We make a few observations about how the software is presented in the article and used in the computational method.

Paper descriptions of conceptual steps contain very limited information about how they map to software. Given the capabilities of the software used, conceptual steps may be mapped to several implemented steps, and vice versa. For example, Polyphred and Polyscan are two separate steps, both implement variant calling but the former does SNP calling only and the latter implements indel and SNP calling steps. Therefore, a requirement in designing a workflow is that it must make clear how each step is implemented in software.

Paper descriptions of software contain limited information about what conceptual steps they implement. For example [TCGA 2008] says: "Putative variants were identified using Polyphred 6.1, Polyscan 3.0, SNPdetector 3, and SNP Compare." Four pieces of software are mentioned, but there are no details that specify what types of variants are detected by each of them. Therefore, in designing a workflow, the mapping of conceptual steps to software must be clearly stated.

A given function can be implemented by many software packages. There are many software packages that provide a desired functionality. As a result, identical functions in different methods may be implemented by different software, making it hard for a scientist to compare workflows. For example, in the workflow in Figure 1 the Consed software is used to perform the genome assembly step. In [Zhang et al 2014], Tophat2 performs this genome assembly step. Therefore, a requirement is that the software steps be described according to their functionality, so that the methods for several papers can be more easily compared by a scientist. Functions should be specified for

each method step so that the correspondences across different software implementations for the same conceptual step will be explicit.

A given software package has many functions. In comparing software to the workflows built from them, we found that many scientific software packages have a large number of functions. Though it is useful for scientists to have multiple functions in one software package, in research papers it can be difficult to tell what software packages are being used for what functions. Sometimes the functionality of a software package is quite broad. For example, the SAMtools software package, used in [Zhang et al 2014], can be used for Variant Calling and Variant Filtering but the article does not explicitly indicate for what function it is used. Therefore, when specifying what software is used to implement a step in a method, it is vital to indicate the specific function of that software to make it unambiguous what conceptual function the software is implementing.

A computational step may perform a data reformatting, conversion, or other minor step that is not conceptually important and therefore is not mentioned in the article. Without a description of these steps, it may not be possible to interpret the results appropriately or to reproduce the method.

In summary, the descriptions of method steps and their implementation in software that are typically found in scientific articles are very ambiguous and incomplete. Computational workflows can eliminate this ambiguity, but they must be intentionally designed to be unambiguous and complete.

3.3 Data Descriptions

Like software, data is described in scientific papers with a mixture of high-level concepts and low-level format references.

Data is often described based on its format rather than its contents. We saw examples of this in the earlier article excerpts. Therefore, a requirement in the design of workflows is that data abstractions should be used to complement step abstractions.

Data formats are sometimes used when data is generated in idiosyncratic formats by specific software used. This can be seen in Figure 1. The Phred software generates output in a format called phd, and as a result the workflow indicates phd_File which is specific to Phred. Thus, a user of the workflow unfamiliar with Phred would find it hard to understand that format. Therefore, a requirement in describing software steps and data in a workflow is to represent explicitly what formats are imposed by the use of specific software packages.

Data of the same type can play very different roles in a method. In the workflow in Figure 1, the varAnnotParams input is an annotated variant parameter file but this type is not represented explicitly. Moreover, it is also not the only annotated file that is input to this method, but is the only one with a name that mentions annotation. Therefore, a requirement is to describe data conceptually according to the type of data contained, and that different data used or generated in the workflow be related by those types.

Data results of the same type may be combined, filtered, or sorted in ways that are not considered important to mention in the paper. Readers must hypothesize these data manipulations.

3.4 Discussion

Through examples we have illustrated that the text descriptions of methods sections of articles makes them hard to reconstruct and replicate into an unambiguous and complete workflow. This is because papers describe methods in a mix of high-level conceptual terms together with mentions of specific software and formats. This makes it hard to understand and compare methods. Another observation is that different readers might be interested in different descriptions of the methods, some more abstract and some more specific. For example, a developer would be interested in data formats and software versions, while a biologist would be more interested in the overall statistical approach used.

Ideally, method descriptions would make clear distinctions between high-level conceptual terminology and implementation terms, both for software and for data. In addition, method descriptions would make it clear what function each step performs, and whether a given function is implemented by a single step or by a set of steps. These desiderata lead us to propose workflow abstractions for describing computational experiments in a paper.

4 WORKFLOW ABSTRACTIONS

A computational method is typically described in terms of the specific software, data, and formats used. However, there are many ways to describe a method conceptually. This section describes different ways to design workflow abstractions that would be useful to make methods more understandable and comparable. Table 1 summarizes the issues identified earlier and the corresponding proposed abstractions to address them.

Table 1. An overview of the issues identified in the papers analyzed and abstractions proposed to address them.

Issue identified	**Abstraction approach**
1) Method descriptions in scientific articles mix mentions of software, data formats, and scientific descriptions of the experiment	Step abstractions
2) Paper descriptions of conceptual steps contain very limited information about how they map to software	Sub-workflow and step abstractions
3) Paper descriptions of software contain limited information about what conceptual steps they implement	Sub-workflow and step abstractions
4) A given function can be implemented by many software packages	Step abstractions
5) A given software package has many functions	Step abstractions
6) A computational step may perform a data reformatting, conversion, or other minor step that is not conceptually important	Criticality abstractions
7) Data is often described based on its format rather than its contents	Data abstractions
8) Data formats are sometimes used when data is generated in idiosyncratic formats by the specific software used	Data abstractions
9) Data of the same type can play very different roles in a method	Data abstractions
10) Data results of the same type may be combined, filtered, or sorted in ways that are not considered important to mention in the paper	Criticality abstractions

4.1 Step Abstractions

A computational workflow can be described at a conceptual level in terms of the functions that each step carries out. Figure 2 describes the same workflow introduced in Figure 1. While Figure 1 describes the software implementation of each step, Figure 2 characterizes the function of each step.

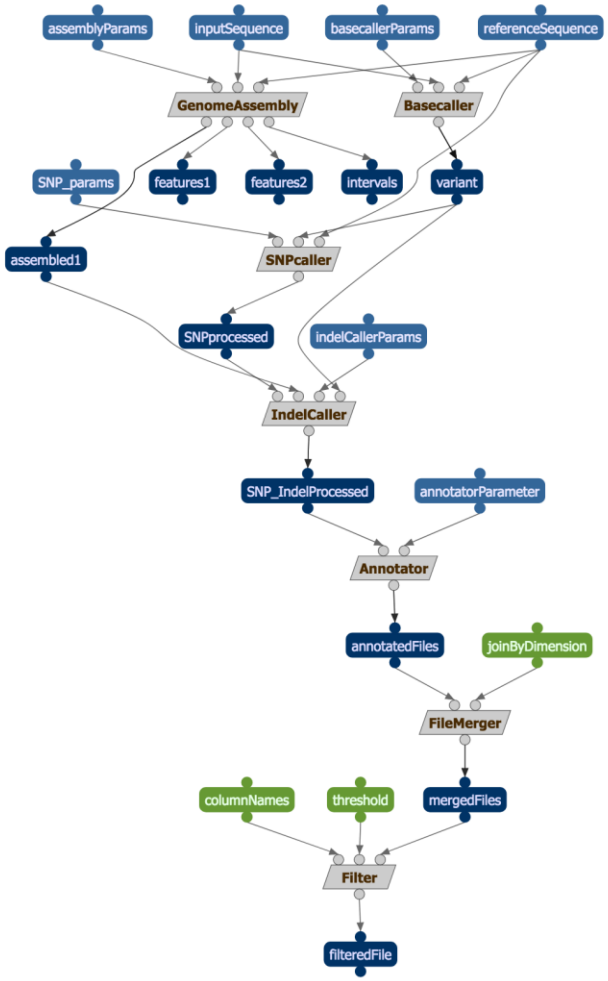

Figure 2: A computational workflow that corresponds to the workflow in Figure 1 but where each step is described conceptually.

At the same time, the software steps in Figure 1 and the conceptual steps of Figure 2 should be mapped to one another. This can be done through a *hierarchy of component functions*, which defines many conceptual functions at different levels of detail. The hierarchy bottoms out with mentions of software that implements the parent function. Note that there may be several software implementations of the same abstract function.

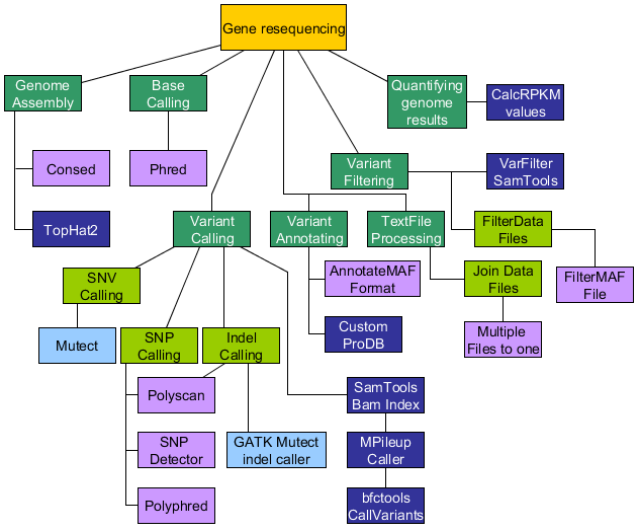

Figure 3: A hierarchy of component functions to describe method steps. The software steps in Figure 1 are shown in dark blue, and the abstract steps in Figure 2 are shown in green, both from [TCGA 2008]. Additional steps in [Imielinski et al 2012] are shown in purple, and those in [Zhang et al 2014] are shown in light blue.

Figure 3 shows a hierarchy of component functions for the steps in Figures 1 and 2. The function in a given node represents a more specific function than the function of its parent node. It also includes steps for the other two articles that we analyzed. Using this hierarchy, it becomes possible to relate the method steps of the three articles.

When designing a workflow, two distinct types of workflows should be created. One type of workflow is an *abstract workflow*, with abstract components that correspond to the more general functions in the hierarchy. These abstract workflows capture the general functionality of methods, and they would be independent of the software used to implement it. A second type of workflow would be a *grounded workflow*, which would specify what software is used to implement each step.

We find that in practice it is hard to create a complete hierarchy of component functions before creating the workflows. We recommend an iterative process, where an initial hierarchy is created and then refined as the workflows are fleshed out.

Depending on the depth of the hierarchy of component functions there could be several abstract workflows that could have different levels of detail and generality. Each abstract workflow may be useful to a different reader, depending on the level of detail that they are looking to find. At the same time, if the workflow contains descriptions of the steps that are too general, it may not be very helpful to a reader. Workflow designers should design appropriate conceptual levels.

A hierarchy of component functions becomes a powerful enabler for automation. Given a concrete workflow, the hierarchy could be used to generate abstract workflows automatically. Conversely, given an abstract workflow, the hierarchy could be used to specialize it and create a concrete workflow. [Gil et al 2011] describe algorithms to do this kind of automation.

4.2 Sub-Workflows

Several components may implement different aspects of the same function. For example, in the workflow of Figures 1 and 2 the Polyphred software and the Polyscan software implement SNP calling and indel calling respectively, which are two aspects of variant calling. The software Annotate_MafFormat annotates the resulting variants with respect to reference genomes. All three steps could be considered as a sub-workflow, with an overarching abstract function of detecting and annotating variants.

A *knowledge base of sub-workflows* would capture these functional decompositions. A sub-workflow would consist of a *root abstract component*, which indicates the overarching abstract function, and a *workflow fragment* that decomposes that function into a set of components at a lower level of abstraction and the dataflow among them. Data abstractions should be taken into account as well as the sub-workflows express functions of different abstraction levels. We discuss data abstractions below.

When designing a workflow, steps that are functionally related should be organized as sub-workflows. There may be alternative ways to group steps in a workflow. Workflow designers should make decisions based on the expected use of the sub-workflow decompositions by readers. The knowledge base of sub-workflows could be dynamically extended based on a growing corpus of workflows created by users. [Garijo et al 2014c] describe techniques to detect workflow fragments automatically.

4.3 Criticality

Some steps in a workflow perform functions that are critical to the overall computational method, while other steps carry out minor format conversions and other ancillary functions. For example, the workflow in Figures 1 and 2 has a step to merge several files. Other workflows have reformatting steps, unit conversion steps, and other functions that manage the details of how the data is

implemented. When describing a method in a paper, these ancillary functions are rarely mentioned. There may be different degrees of criticality, depending on how much detail each reader is interested in seeing.

This kind of abstraction could be captured in a *hierarchy of criticality levels.* This hierarchy would identify the importance of including a step in a scientific description of a method. [Garijo et al 2014b] describe an approach to identifying criticality based on a library of workflow motifs that include data pre-processing, visualization, and format conversion. Criticality levels are highly dependent on the specific domain, but a broad methodology to design those categories could be more generally designed.

4.4 Data Abstractions

Data type abstractions should be included in all three hierarchies above. The data type in a node would represent data that is of a more specific type than its parent node, for example because it is of a subtype or has more specific metadata properties. In the hierarchy of component functions, each abstract component function should specify inputs and outputs in terms of those general types. At the bottom of the hierarchy, a component is specified with a specific software invocation, including the exact command line call to invoke the software and all the input data types and formats that the software expects. In the hierarchy of sub-workflows, the root component may refer to data types that are more abstract than those of the workflow fragment.

Data abstractions depend on the domain. In multi-omics, there are many aspects of data that can be described in very specific terms but can be abstracted away when describing an experiment in scientific terms. Characteristics of a dataset that can lead to useful data abstractions include: 1) type of sequence, such as RNA, DNA, etc.; 2) annotations on those sequences, such as indels, CNVs, SNPs, etc.; 3) formats that are often imposed by how software works, such as FASTA, MAF, phd, etc.; 4) level of detail or accuracy on the sequences, for example sequences obtained with next-generation sequencing machines are more accurate; 5) the role of a dataset for a specific component, for example a sequence can be a patient sequence or a reference sequence.

Workflow designers should create a *taxonomy of data abstractions* that facilitate the abstractions needed for the three hierarchies discussed earlier. In our work, we have found that a proliferation of data types makes the creation of workflows more complex. Instead, we create properties for describing the different characteristics of data.

5 CONCLUSIONS

This paper motivates the need for capturing abstractions in the design scientific workflows. These abstractions are based on our analysis of published articles and the workflows created to reconstruct their methods. The proposed abstractions are captured in hierarchies of component functions and criticality as well as knowledge bases of sub-workflows, and need to be supported by data abstractions. Using these abstractions, different workflows can be created to describe the same computation for readers with different interests. In future work, we plan to develop these abstractions for a target domain and associated publications, in order to demonstrate their benefits.

Acknowledgements. We gratefully acknowledge support from the Defense Advanced Research Projects Agency through the SIMPLEX program with award W911NF-15-1-0555, the National Institutes of Health with awards 1U01CA196387 and 1R01GM117097, and the Canary Foundation.

REFERENCES

[Baggerly and Coombes 2009] Baggerly KA, and KR Coombes. "Deriving Chemosensitivity from Cell Lines: Forensic Bioinformatics and Reproducible Research in High-Throughput Biology." Annals of Applied Statistics 3 (4), 2009.

[Donoho et al 2009] Donoho DL, Maleki A, Rahman IU, Shahram M, and V Stodden. "Reproducible Research in Computational Harmonic Analysis." Computing in Science & Engineering 11 (1): 8–18, 2009.

[Garijo et al 2013] Garijo D, Kinnings S, Xie L, Xie L, Zhang Y, Bourne PE, and Y Gil. "Quantifying Reproducibility in Computational Biology: The Case of the Tuberculosis Drugome." PLoS ONE 8 (11), 2013.

[Garijo et al 2014a] Garijo D, Corcho O, Gil Y, Braskie MN, Hibar D, Hua X, Jahanshad N, Thompson P, and Toga AW. "Workflow Reuse in Practice: A Study of Neuroimaging Pipeline Users." Proceedings of the 10th IEEE International Conference on e-Science, 2014.

[Garijo et al 2014b] Garijo D, Alper P, Belhajjame K, Corcho O, Gil Y, and C Goble. "Common Motifs in Scientific Workflows: An Empirical Analysis." Future Generation Computer Systems 36, 2014.

[Garijo et al 2014c] Garijo D, Corcho O, Gil Y, Gutman BA, Dinov ID, Thompson P, and AW Toga. 2014. "FragFlow: Automated Fragment Detection in Scientific

Workflows." Proceedings of the 10th IEEE International Conference on e-Science, 2014.

[Garijo et al 2017] Garijo D, Gil Y, and O Corcho. "Abstract, Link, Publish, Exploit: An End to End Framework for Workflow Sharing." Future Generation Computer Systems, 2017.

[Gil 2015] Gil, Y. "Human Tutorial Instruction in the Raw." ACM Transactions on Interactive Intelligent Systems, 5 (1): 1–29, 2015.

[Gil and Garijo 2017] Gil Y, and D Garijo. "Towards Automating Data Narratives." Proceedings of the ACM Conf. on Intelligent User Interfaces, 2017.

[Gil et al 2011] Gil Y, Gonzalez-Calero PA, Kim J, Moody J, and V. Ratnakar. "A Semantic Framework for Automatic Generation of Computational Workflows Using Distributed Data and Component Catalogs." Journal of Experimental and Theoretical Artificial Intelligence, 23(4), 2011.

[Groth and Gil 2009] Groth P and Y Gil. "Analyzing the Gap between Workflows and Their Natural Language Descriptions." Proceedings of the IEEE International Workshop on Scientific Workflows (SWF), 2009.

[Knoblock 2017] Knoblock M. "Designing Useful Abstractions for Multi-Omics Data Analysis." Technical Report, Information Sciences Institute, University of Southern California, October 2017.

[Imielinski et al 2012] Imielinski M, Berger AH, Hammerman PS, Hernandez B, et al. "Mapping the hallmarks of lung adenocarcinoma with massively parallel sequencing." Cell;150(6):1107-20, 2012.

[Ince et al 2012] Ince DC, Hatton L, and J Graham-Cumming. "The Case for Open Computer Programs." Nature, Vol 482, 2012.

[Ioannidis et al 2009] Ioannidis JPA, Allison DB, et al. "Repeatability of Published Microarray Gene Expression Analyses." Nature Genetics 41 (2), 2009.

[TCGA 2008] The Cancer Genome Atlas (TCGA) collaboration. "Comprehensive Genomic Characterization Defines Human Glioblastoma Genes and Core Pathways". Nature, 455, 1061-1068, 23 October 2008.

[Stodden et al 2016] Stodden V, McNutt M, Bailey DH, Deelman E, Gil Y, Hanson B, Heroux MA, Ioannidis JP, and M Taufer. "Enhancing Reproducibility for Computational Methods." Science, 354, 2016.

[Steehouder et al 2000] Steehouder, M., Karreman, J. and Ummelen, N. Making sense of step-by-step procedures. Proceedings of 2000 Joint IEEE International and 18th Annual Conference on Computer Documentation (IPCC/SIGDOC 2000)

[Taylor et al 2006] Taylor IJ, Deelman E, Gannon DB, and M Shields. "Workflows for e-Science: scientific workflows for grids." Springer, 2006.

[Van Noorden 2015] Van Noorden, R. Sluggish data sharing hampers reproducibility effort. Nature, 2015.

[Xie 2015] Y Xie. "Dynamic Documents with R and knitr." CRC Press, 2015.

[Zhang et al 2014] Zhang B, Wang J, Wang X, et al. "Proteogenomic Characterization of Human Colon and Rectal Cancer." Nature 513 (7518): 382–87, 2014.