NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance

Lucas A. M. C. Carvalho[1], Regina Wang[2], Yolanda Gil[2], Daniel Garijo[2]
[1]University of Campinas, Institute of Computing, Campinas, SP, Brazil
[2]University of Southern California, Information Sciences Institute, Marina del Rey, CA, U.S.A

lucas.carvalho@ic.unicamp.br, gil@isi.edu, dgarijo@isi.edu

ABSTRACT

Interactive notebooks are increasingly popular among scientists to expose computational methods and share their results, but it may be challenging to track the dataflow of their outcome, and therefore their provenance. This paper presents an approach to convert notebooks into scientific workflows, which capture explicitly the dataflow across software components and facilitates tracking provenance of new results. Users can first manually rewrite notebooks according to well-specified guidelines, and then use an automated tool to generate workflows from the modified notebooks. Our approach is implemented in NiW (Notebooks into Workflows), and we demonstrate how to use it by generating workflows with third-party notebooks. The resulting workflows have explicit dataflow, which facilitates tracking provenance of new results, comparison of workflows, and subworkflow mining. Our guidelines can also be used to improve understandability of notebooks by making the dataflow more explicit.

KEYWORDS

Scientific Workflows; Workflows; Electronic Notebooks.

1 INTRODUCTION

Interactive notebooks have become very popular in science to capture computational experiments [Shen 2014]. These notebooks include code, visualizations, and explanations, and can be easily shared and re-run,

As scientists carry out their research, they may need to compare the results and methods of different experiments. This involves comparing final results, comparing intermediate results, comparing steps of the method, and comparing parameter values. Since notebooks contain raw code, it can be hard to understand how new results are generated, as well as to compare notebooks. In contrast, workflows offer modular components to run code, and have an explicit dataflow. This can facilitate provenance capture, as well as automated mining of reusable workflow fragments [Garijo et al 2014a]. Workflows also facilitates understanding, particularly for non-programmers [Garijo et al 2014b].

This paper presents an approach for converting notebooks into workflows by mapping various aspects of notebook cells into workflow components and dataflow. Our approach is implemented in NiW, a prototype tool to convert Jupyter Notebooks[1] into WINGS workflows [Gil et al 2011]. We illustrate the use of NiW with some exemplary notebooks. Based on the assumptions of our approach, we propose a set of guidelines for designing notebooks that facilitate the conversion and can be used by notebook developers to improve the understandability of their notebooks.

2 DATAFLOW AND PROVENANCE IN WORKFLOWS AND NOTEBOOKS

This section discusses general issues for identifying dataflow and tracking provenance in notebooks, compared with simple dataflow in workflows. Our work to date has focused on mapping Jupyter Notebooks to workflows that can be used in the WINGS workflow system [Gil et al 2011], but many of the issues will be common for other notebook and workflow systems.

2.1 Simple Dataflow in Workflows

Workflows capture explicitly the dataflow across software components. We describe here a very simple dataflow representation and workflow structure that we assume in the rest of the paper. This approach is used in several workflow systems, including WINGS [Gil et al 2011], Pegasus/Condor [Deelman et al 2005], and Apache Taverna [Oinn et al 2006].

Each software *component* (or *step*) of the workflow may have multiple datasets as *inputs*, multiple datasets as *outputs*, and *parameters*, which are provided as simple numeric or Boolean values. A dataset generated by a component can be input to another component, thereby indicating the flow of data (i.e., the dataflow) from a component to another.

A workflow management system can run a workflow if the software components can be executed and the respective input datasets and parameter values are provided. Because the dataflow is explicitly captured in the workflow, the system can record the provenance of each new dataset generated by the workflow.

The dataflow of a workflow is often shown as a graph. Workflows can be compared as graphs. Indeed, graph algorithms have been used to query workflow repositories [Bergmann and Gil 2014], and to mine workflow repositories to find commonly occurring subworkflows [Garijo et al 2014a]. Visual user interfaces that show the dataflow graph in a workflow are easy to use for non-programmers [Hauder et al 2011].

[1] http://jupyter.org

2.2 Overview of Computational Notebooks

Notebooks aggregate text and code, grouped into a sequence of containers or *cells*. Cells can be *code* cells, *raw* cells, and *markdown* cells. Code Cells have running code usually with one programming language such as Python, PHP, Java, etc. The code cells are the heart of notebooks. Markdown cells are comments and documentation, written in HTML so users can add pictures, formatting, etc. These cells are not linked with any other cells and run without interfering other cells. The raw cells are much less commonly used since they do not run and markdown cells serve better comments than them. Unlike raw and markdown cells, code cells are linked with each other and when the notebook is run, the code cells run like a chunk of code. Thus, when a previous code cell does something, the next code cell carries it on as though there were no other cells in between. Outputs are shown in the notebook when a cell code includes a plot or print statement.

The Jupyter Notebook[2] is one of the most popular notebook platforms. They were originally named IPython Notebooks since they are primarily used with Python, but expanded since people have written kernels for several other programming languages other than Python. A kernel is a program that runs and introspects the user's code.

2.3 Understanding Dataflow of Notebooks

We analyzed a diversity of Jupyter Notebooks to understand their dataflow and the provenance of their results.

Our focus has been a collection of related hydrology notebooks[3]. The notebooks use MODFLOW[4], a widely-used groundwater model created by USGS (U.S. Geological Survey) and written in Fortran. The notebooks use a core FloPy package[5] to run MODFLOW from Python, updating the file structure, writing new input files, running the model and exporting the inputs and outputs. Tracking the provenance of the results is hard, since each invokes FloPy and MODFLOW differently. As new versions of MODFLOW and FloPy are released, tracking the provenance and comparing the notebooks will become more challenging.

Common problems that we found include:

1. **Processing:** A user may not have a clear understanding of what are the main processing cells of a notebook. For example, some of the cells have only one line of code, while other cells have lines of code only to assign values to variables. This basically indicates that those cells are not processing units and should be placed in the same cell that use those variables.
2. **Dataflow:** Input file names are either implicit in the code or defined as parameters through method calls in previous cells. This may cause an implicit dependency between cells, and therefore make it difficult to understand the dataflow from different cells. In addition, users may also have difficulty figuring out what files were generated by a given cell.
3. **Inputs:** Input files may contain pointers to other files that are opened and used as inputs inside of a cell. This creates an implicit dependency that is difficult to detect.
4. **Outputs:** When a cell does not have an explicit output is very difficult to understand what kind of process that cell performed. A cell may overwrite a file with the same name generated by other cells, so it can be hard to track the provenance of newly generated files. Notebooks can generate visualizations, but those do not necessarily generate output files.
5. **Files:** It is very difficult to understand how the files in notebook folders correspond to the cells that used or generated the files.
6. **Data:** Some notebooks are available in repositories without any test data. Therefore, this makes it hard to understand the expected data format of the input files and the outputs generated by the notebook.

In summary, many problems may arise in trying to understand what is the dataflow across the cells of a notebook and how they use or generate notebook files. This makes it very hard to figure out the provenance of any results. This also makes it hard to understand a notebook, as well as comparing different notebooks.

3 MAPPING NOTEBOOKS TO WORKFLOWS

When mapping notebooks into workflows, many issues must be addressed. We discuss here those issues, and our approach to address them. We start with general issues. After that, we focus on issues specific to Jupyter Notebooks and Python, since that has been the focus of our work so far. Then we discuss issues specific to our WINGS workflow system, which is the target of our mappings.

3. 1 Components

3.1.1 Executable Code

Differences: Each component in a workflow must have some running code within it. In notebooks, cells may contain solely value assignment to variables, function declarations or library imports, or documentation, which are not executable code by themselves and cannot be easily mapped into a workflow component. Another difference is that in notebooks even though code is split cell by cell, most of the splits only exist to benefit human readability and do not actually affect the code itself. Cells are used just to modularize the code; thus, different users may break up the code at different places and it will not matter much. In contrast, in workflows the code is split into components which are isolated from one another. An example of this is that if a variable is created in a component and used in a second component, the latter will not have access to this variable unless it is generated by the former and explicitly consumed by the second component.

[2] http://jupyter.org/
[3] https://github.com/modflowpy/flopy/tree/develop/examples/Notebooks
[4] https://water.usgs.gov/ogw/modflow/
[5] https://github.com/modflowpy/flopy

Approach: Each workflow component will correspond to one notebook cell with running code. If a notebook cell does not have running code and only has library imports or method declarations, it will become part of a cell that requires that information.

3.1.2 Libraries and Methods

Differences: A notebook only needs to import a library or state a method once. Since workflows are componentized, the imports and method declarations need to be done in each workflow component that uses them.

Approach: Every library used in the notebook will be imported into each workflow component. A method will be included in a component only if the method is to be used in it.

3.1.3 Open Files

Differences: A notebook can open a file and use it in any subsequent cell. In a workflow, a file can be used only inside the component that has that file as input.

Approach: If a file is opened and used across many cells, those cells will be merged into a single component. Note that an alternative approach might be to create separate components for each of the cells and open and close the file in each of those components, but this would result in inefficiencies if the data is written to files and read from files too many times.

3.1.4 Markdown Cells

Differences: Markdown cells in the notebooks do not contain running code, but need to be included in the workflow as documentation so that the information that they contain is not lost. Workflow components can have documentation. There can be more markdown cells than code cells. In addition, the relationship between markdown cells and code cell is not explicit. A markdown cell may be related to either its previous or its subsequent cell.

Approach: Since the relationship between markdown cells code cells is unknown, the assignment is made in an arbitrary way. A markdown cells' information will be attached to the documentation of the component created for its subsequent code cell.

3.1.4 Component Naming

Differences: In a notebook, a cell does not have a name. In a workflow, a component has a name that generally describes the function of the component is in the workflow.

Approach: A name will be generated for each component of the workflow, starting with "Component" followed by the ordering number from the cell (e.g., Component1, Component2 and Component3).

3.2 Dataflow

3.2.1 Parameters

Differences: In notebooks method parameters are set through program variables. In workflows, parameters are inputs to components and provided by users. In workflows, if parameters are coming from other components, these parameters must be passed explicitly through a file.

Approach: Variables of primitive types (i.e., boolean, string, integer, float, date, etc) in notebooks will be mapped to parameters in workflows, and they will be given the name that was used in the notebook.

3.2.2 Input Files

Differences: A notebook may be given input data once at the beginning, and there is no need to pass data through files from cell to cell. In a workflow, a component must output a data file that is then an input to another component.

Approach: A data file will be explicitly generated from the notebook code in order to be passed to another component in the workflow. Code will be added to the component that generates the data so that the resulting data is written into a file that can be passed to the next component.

3.2.3 Workflow Structure

Differences: In a notebook, cells are specified sequentially. In a workflow, the flow of data among components must be specified, and is not a sequential flow.

Approach: The identifiers of the files generated and consumed by components will be used to obtain the dataflow between the components, and the dataflow will be explicitly stated in a workflow structure.

3.3 Python-Specific Differences

A few mappings are challenging because of specific way that Python is used in Jupyter Notebooks. The IPython kernel allows notebooks to use special functions that the standard Python interpreter does not support. Since a workflow component would be executed using standard Python, these functions cannot be directly mapped. Notebooks are also designed for human readability and are, as a result, much more documented and aggregate more resources than plain Python code. In addition, notebooks include Python commands to generate visualizations (e.g., graphs), which are executed and the results shown in the notebook but not necessarily saved.

3.3.1 Visualizations

Differences: Notebooks show visualizations which may not be saved into a file. Workflow components would generate visualizations and save them in an output file.

Approach: If the notebook does not save a visualization, the workflow will automatically save the visualization in an output file.

3.3.2 Magic Commands

Differences: Notebooks have IPython kernel built-in commands know as *magic commands*. They start with "%" and can list all variables, return the directory being used, etc. Magic commands only work on IPython kernels. The Python standard interpreter does not recognize these commands since it is not Python code.

Approach: Magic commands, will be replaced with Python code that implements them. Other magic commands that may appear in a notebook will not be mapped.

3.3.3 Automatic Output

Differences: In a notebook, if a variable is printed (using the print statement), it will appear as an output. In a software component, that code would not generate an output.

Approach: A workflow component that includes print commands will have an extra output with all the results from the print statements.

3.4 WINGS-Specific Differences

In WINGS, the code for each software component has an associated script that indicates the command line invocation for the software. The script may also set parameter values. Notebooks do not have this. Input files are also treated differently from notebooks. In WINGS, input files are classified into a hierarchy of data types. Semantic metadata can be specified as well for all input files. Notebooks do not have either.

3.4.1 Software Components

Differences: The script for a workflow component in WINGS must specify the number of inputs, parameters, and outputs that the code for the component expects. Notebooks do not have this information.

Approach: After mapping the code from the cells of the notebook into components, a script will be generated for each component indicating the invocation command together with the total number of inputs, outputs, and parameters.

3.4.2 Input Data Files

Differences: In WINGS, each input file is assigned a data type. In notebooks, there are no data types or metadata for the files.

Approach: All input files will be considered to be of the same general data type.

3.4.3 Workflow Components

Differences: In a notebook, code is specified directly in cells. In WINGS, a workflow component must be created and given a type from a component type hierarchy. The inputs, outputs, and parameters must be specified. Finally, the workflow component must be associated with the code in the software component (see 3.4.1 above).

Approach: All workflow components will be given a general component type. Their inputs, outputs, and parameters will be specified based on the script of the associated software component.

3.5 Usability Requirements

Our approach requires that users make changes to their notebooks in order to facilitate the conversion of notebooks into workflows. We took into account additional requirements to reduce the burden to users and maximize the utility of the changes required:

- The user should have to make minimal changes to a notebook to allow the conversion tool to generate a workflow.
- Any changes made to a notebook should improve its readability and documentation as well as facilitate the conversion.
- Any changes made to a notebook should be independent of the target workflow system.
- Any changes made to a notebook should improve the understanding of the dataflow.
- The workflows should include all the documentation of the original notebooks.
- As many results generated by a notebook should be generated in the workflow as well, even if they are not explicit in the notebook.
- A conversion tool should automate the process as much as possible, and some manual intervention may be needed after running it.

4 GUIDELINES TO DEVELOP NOTEBOOKS

Based on our approach to map notebooks into workflows, we designed a set of guidelines that users can follow to facilitate the conversion of notebooks into workflows. Even if workflows are not generated, by following these guidelines users will create notebooks that have more explicit dataflow, which will facilitate understanding, comparisons, and reuse by others.

We list here the set of guidelines, each with a justification and an example when necessary.

1. **Provide at least one cell with running code:** a workflow component must have running code within it to be created and a workflow must be composed of at least one component.
2. **Write into files any newly generated data:** the code in a cell should write to files with the data generated, so that other cells can use those files. This will make the dataflow across cells more clear.
3. **Close files after using them:** if files are opened and used across many cells, all those cells should be merged into a single component, making the workflow components more modular.
4. **Do not use variables as file names to open files:** this is due to the fact that is difficult to track the variable values across components and our tool does not track them so far.
5. **Provide meaningful names for files:** these names should make clear what kind of data the files contain. Avoid names such as "load", "data05", "experiment-data". Instead, use names like "SensorReadings" or "PluviometricCalculation". This makes the visual presentation of the workflow more readable.

These guidelines aim to facilitate the automated conversion of notebooks to workflows. They will also improve the understandability of notebooks by making the dataflow more clear.

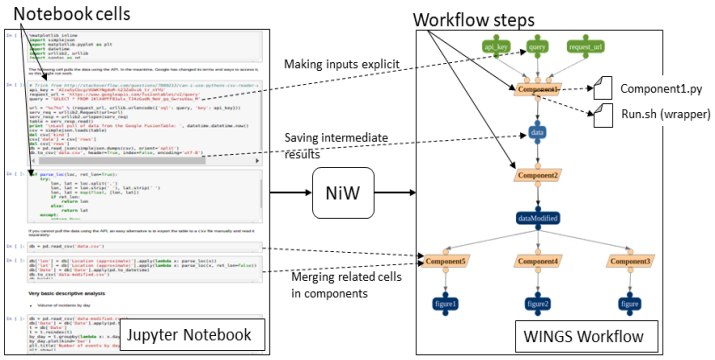

Figure 1. Using NiW to generate a WINGS workflow from a Jupyter Notebook.

5 NiW: A Tool for Converting Notebooks into Workflows

We implemented our approach in NiW (Notebooks into Workflows). Our current NiW prototype creates workflows for WINGS system from Jupyter Notebooks. The prototype code is available at https://w3id.org/niw.

NiW takes as input a notebook file and generates: 1) a zip file for each workflow component (e.g., Component1.zip), containing the component code as a Python script (e.g., Component1.py), a script file (named io.sh) to handle the inputs and outputs of the component, and a script (named run) to execute the component; 2) the name of the inputs, outputs, and parameters for each component; and 3) the WINGS workflow. NiW takes the files (1) and (2) to create (3) the WINGS workflow, automatically. The prototype also creates the data type named File in the corresponding WINGS domain, if it does not exist, and associate all data files to this data type. NiW uses the notebook filename to name the workflow.

Figure 1 illustrates how the notebooks are converted into workflows by NiW using the approach outlined in Section 3.

5.1 Current Limitations of NiW

The following are limitations of our current NiW implementation. Python is the only programming language supported in cells containing running code. The use of magic commands is restricted, currently only the magic command %matplotlib (which allows visualizations to be generated) is supported. The only methods supported for opening files are the built-in method "open" and the method "read_csv" from Pandas[6], a well-known data analysis library in Python. Only Matplotlib can be used to generate visualizations. Finally, the notebook should run fully without errors: If an error occurs while executing a notebook, it would be difficult to identify how data are generated and used throughout all the cells.

5.2 Using NiW

To demonstrate how NiW works we have chosen a Jupyter Notebook and coded in Python, taken from http://nbviewer.jupyter.org/gist/darribas/4121857. This notebook uses real world data provided by The Guardian newspaper to analyze and map the incidents during the 2012 Gaza-Israel crisis, exploiting the spatial as well as the temporal dimension of the data. The modified versions of the notebook, the WINGS workflow, and the workflow execution are available at https://w3id.org/niw/papers/sciknow2017.

We modified the notebook based on the guidelines presented in Section 4, and to address the limitations of our current implementation of NiW mentioned above. The changes required by our guidelines in the notebook code were related to guideline #3 – to write newly generated data into files: (1) saving the data retrieved online in a local file, instead of loading it in memory to be used in subsequent cells; (2) saving changes made to the data in each cell into a new file; (3) opening the updated data file saved by (2) in subsequent cells.

The nine code cells in the original notebook were merged to generate five workflow components. The code cells containing only library imports were merged with other components as well as the cells containing the declaration code of the function *parse_loc*. The inputs of the workflow are the parameters *api_key*, *request_url* and *query*, variables with assignment to string values in the original cell. After retrieving the data, it is saved as a CSV file by *Component1*. Components 3, 4 and 5 save the graphs generated for future inspection, originally showed inline in the notebook.

Figure 2 shows the workflow created by NiW using the modified notebook as input. The modified notebook is improved for use by scientists with respect to the original version in several respects, such as making inputs explicit, saving intermediate results, merging related cells into components, and making outputs explicit.

One benefit of the workflow is to support comparison and provenance when run with new datasets. Journalists from The Guardian created the input dataset in collaboration with users from the Internet. If the input data changes, the workflow could easily be executed again, and its execution results can be compared. Since all the intermediate results are stored as provenance information, they may also be compared. Another benefit of the workflow is to compare the results when the code changes. In this case, the notebook is collaborative and can be extended by users via GitHub. When the notebook is changed,

[6] http://pandas.pydata.org

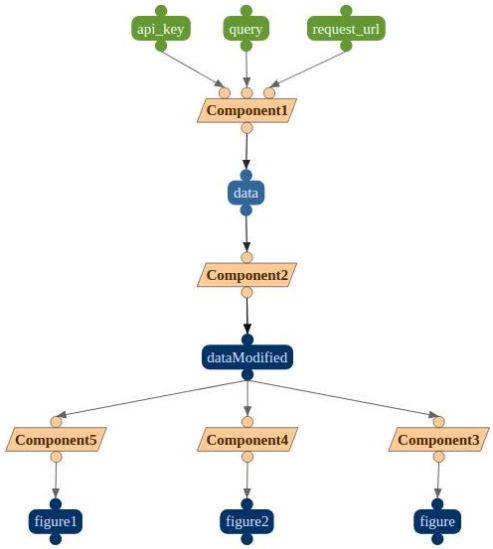

Figure 2. WINGS workflow created from the Guardian notebook.

NiW can be re-run and a new workflow would be generated and executed. The workflow executions can be compared very easily.

6 RELATED WORK

There are several related approaches that aim to expose the dataflow within scripts and/or to map scripts into structures that support provenance tracking.

NoWorkflow [Murta et al 2014] captures provenance information from scripts to help scientists understanding the script execution. However, this approach does not simplify the understanding of the script specifications to non-programmers.

YesWorkflow [McPhillips et al 2015] enables scientists to make explicit the dataflow in scripts by providing special tags that scientists use to annotate the scripts. These annotations split the script into steps and clarify the inputs and outputs of each step as well as the structure of the workflow. It enables the creation of a visualization based on these annotations, helping scientists to understand the dataflow within the script. However, a scientist still has a script which is difficult to repurpose compared to workflows.

W2Share [Carvalho et al 2017] focus on the conversion of scripts into scientific workflows. This approach automatically generates workflows from annotated scripts. However, this methodology lacks a clear definition of the structure of scripts that it covers neither proposes guidelines to change the scripts to this expected structure before converting them. Moreover, this work does not consider particularities present in notebooks.

[Pimentel et al 2015] proposes an approach to capture provenance from notebooks. They build upon noWorkflow to capture provenance automatically allowing the analysis of provenance information within the notebook, both to reason about and to debug their work. In our work, we designed guidelines to improve the understanding of the notebooks by scientists and then converted notebooks into workflows automatically.

7 CONCLUSIONS

We presented an approach to map notebook cells into workflow components, addressing many issues that arise because of the implicit dataflow in notebooks. We introduced a set of general guidelines for notebook developers that help make dataflow more explicit, which can improve understandability and provenance tracking. We implemented NiW, a prototype tool that can convert notebooks that follow those guidelines into workflows, in particular Jupyter Notebooks into WINGS workflows. An interesting direction for future work is to explore the use of the workflows generated for comparing different notebooks.

ACKNOWLEDGMENTS

This work was supported in part by a grant from the US National Science Foundation under award ICER-1440323, and in part by the Sao Paulo Research Foundation (FAPESP) under grants 2017/03570-3, 2014/23861-4 and 2013/08293-7. We would like to thank many collaborators for their feedback on this work, in particular Jeremy White and Zachary Stanko.

REFERENCES

[Bergmann and Gil 2014] Similarity Assessment and Efficient Retrieval of Semantic Workflows. Bergmann, R.; and Gil, Y. Information Systems Journal, 40. 2014.

[Carvalho et al 2017] Implementing W2Share: Supporting Reproducibility and Quality Assessment in eScience. Carvalho, Lucas A. M. C.; Malaverri, J. E. G.; Medeiros, C. B. In Proceedings of the 11th Brazilian e-Science Workshop, São Paulo, Brazil, 2017.

[Deelman et al 2005] "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems." Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman,C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C. and Katz, D. S. (2005) Scientific Programming Journal, vol. 13, pp. 219-237.

[Garijo et al 2014a] FragFlow: Automated Fragment Detection in Scientific Workflows . Garijo, D.; Corcho, O.; Gil, Y.; Gutman, B. A.; Dinov, I. D.; Thompson, P.; and Toga, A. W. In Proceedings of the IEEE Conference on e-Science, Guaruja, Brazil, 2014.

[Garijo et al 2014b] Workflow Reuse in Practice: A Study of Neuroimaging Pipeline Users . Garijo, D.; Corcho, O.; Gil, Y.; Braskie, M. N.; Hibar, D.; Hua, X.; Jahanshad, N.; Thompson, P.; and Toga, A. W. In Proceedings of the IEEE Conference on e-Science, Guarujua, Brazil, 2014.

[Gil et al 2011] Wings: Intelligent Workflow-Based Design of Computational Experiments. Gil, Y.; Ratnakar, V.; Kim, J.; Gonzalez-Calero, P. A.; Groth, P.; Moody, J.; and Deelman, E. IEEE Intelligent Systems, 26(1). 2011.

[Hauder et al 2011] Making Data Analysis Expertise Broadly Accessible through Workflows. Hauder, M.; Gil, Y.; Sethi, R.; Liu, Y.; and Jo, H. In Proceedings of the Sixth Workshop on Workflows in Support of Large-Scale Science (WORKS'11), held in conjunction with SC 2011, Seattle, Washington, 2011.

[McPhillips et al 2015] YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. McPhillips, T., Song, T., Kolisnik, T., Aulenbach, S., Belhajjame, K., Bocinsky, K., Cao, Y., Chirigati, F., Dey, S., Freire, J. and Huntzinger, D. International Journal of Digital Curation 10, no. 1 (2015): 298-313.

[Murta et al 2014] noWorkflow: capturing and analyzing provenance of scripts. Murta, L., Braganholo, V., Chirigati, F., Koop, D. and Freire, J. In *International Provenance and Annotation Workshop* (pp. 71-83). Springer. 2014.

[Oinn et al 2006] "Taverna: lessons in creating a workflow environment for the life sciences." Oinn, T., M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Concurrency and Computation: Practice and Experience, 18(10), 2006.

[Pimentel et al 2015] Collecting and analyzing provenance on interactive notebooks: when IPython meets noWorkflow. Pimentel, J.F.N., Braganholo, V., Murta, L. and Freire, J. In *Workshop on the Theory and Practice of Provenance (TaPP), Edinburgh, Scotland* (pp. 155-167), 2015.

[Shen 2014] Interactive notebooks: Sharing the code. H. Shen. Nature, 05 November 2014.