FileWeaver: Flexible File Management with Automatic Dependency Tracking

Julien Gori, Han Han, Michel Beaudouin-Lafon

FileWeaver: Flexible File Management with Automatic Dependency Tracking

Julien Gori **Han L. Han** **Michel Beaudouin-Lafon**
Université Paris-Saclay, CNRS, Inria,
Laboratoire de Recherche en Informatique
F-91400 Orsay, France
{jgori, han.han, mbl}@lri.fr

ABSTRACT
Knowledge management and sharing involves a variety of specialized but isolated software tools, tied together by the files that these tools use and produce. We interviewed 23 scientists and found that they all had difficulties using the file system to keep track of, re-find and maintain consistency among related but distributed information. We introduce *FileWeaver*, a system that automatically detects dependencies among files without explicit user action, tracks their history, and lets users interact directly with the graphs representing these dependencies and version history. Changes to a file can trigger *recipes*, either automatically or under user control, to keep the file consistent with its dependants. Users can merge variants of a file, e.g. different output formats, into a polymorphic file, or *morph*, and automate the management of these variants. By making dependencies among files explicit and visible, *FileWeaver* facilitates the automation of workflows by scientists and other users who rely on the file system to manage their data.

Author Keywords
File system; Dependency management; Version management; Workflows

CCS Concepts
•**Human-centered computing** → **Graphical user interfaces; Interaction techniques;**

INTRODUCTION
While some users can manage their entire workflow within a single application or integrated application suite, many knowledge workers, such as scientists, must use several specialized tools that are not designed to dialog with each other. For example, Oleksik et al. [24] describes how researchers use a mix of standard office productivity tools and specialized tools for experimental protocols and data analysis, while Zhang et al. [43] list up to 13 different categories of tools used by data scientists.

Proceedings of the 33rd ACM Symposium on User Interface Software and Technology, UIST '20, October 20–23, 2020, Virtual Event, USA
ACM ISBN 978-1-4503-7514-6/20/10 ...$15.00.
http://dx.doi.org/10.1145/3379337.3415830

Specialized tools typically load and save information in proprietary and/or binary data formats, such as Matlab[1] `.mat` files or SPSS[2] `.sav` files. Knowledge workers have to rely on standardized exchange file formats and file format converters to communicate information from one application to the other, leading to a multiplication of files.

Moreover, as exemplified by Guo's "typical" workflow of a data scientist [8, Fig. 2.1], knowledge workers' practices often consist of several iterations of exploratory, production and dissemination phases, in which workers create copies of files to save their work, file revisions, e.g. to revise the logic of their code, and file variants, e.g. to modify parameter values. Jensen et al. [12] observed that short- and long-term pauses can cause users to forget about the organization of their files when they need to access them.

Neither the file system nor file navigation tools are designed to track the *relationships* between files nor the *histories* of files, offering little support to knowledge workers for managing the numerous files and associated workflows that their work requires. Software designed to capture file *provenance* [21, 22] addresses a similar issue by capturing metadata describing the origin of the files as well as all the actions that are performed on it and the actors who perform them. However, we do not know if this fine-grained approach is adapted to the needs of knowledge workers.

In this paper, we first investigate how and with which tools knowledge workers manage their information. We interviewed 23 scientists —an extreme example of knowledge workers— who were not specifically trained in software engineering nor computer science and identified six types of problems they face. Based on these findings, we describe *FileWeaver*, a system that automatically detects dependencies among files, tracks their history, and lets users interact directly with the graphs representing these dependencies and version history. Changes to a file can trigger *recipes*, either automatically or under user control, to keep the file consistent with its dependants. We describe *FileWeaver*'s features and illustrate how they address some of the issues identified in the user study. We conclude with an analysis of *FileWeaver* using Green's cognitive dimensions [7] and directions for future work.

[1] https://mathworks.com/products/matlab.html
[2] https://www.ibm.com/analytics/spss-statistics-software

RELATED WORK

We review previous research in three main areas: the issues related to *information silos*, i.e. the fact that digital information tends to be trapped in proprietary files formats and closed systems; the practices and workflows used by scientists in their data analysis tasks; and the studies and tools addressing the management of file relationships.

Information Silos

Previous work has identified the difficulties that users encounter with the rigid hierarchical organization of current file systems [5, 2, 29] and the information fragmentation created by *information silos* [29, 13]. In a study of users' desktops, Ravasio et al. [29] reported the interviewees' frustration with the fragmentation of information across files and most interviewees expressed the need to have their information linked together. The Placeless system [5] partially addressed these issues by letting users create and manage their own document properties. Karger [13] proposed three approaches to address information fragmentation: grouping (including tagging), annotating, and linking; and showed that linking related information can support users orienteering behavior [41].

Research on scientists' work practices further highlights these issues. Oleksik et al. [24] studied the artifact ecology of a research center and found that scientists used multiple computing applications to create information artifacts that are locked into applications, making it difficult to reuse content and get a unified view of the related research material. Their follow-up study [23] revealed the need to support three types of links: inheritance (source document), spatial proximity (spatial layout), and explicit links (resources and notes). Tabard et al. [39] studied lab notebooks and found that biologists work with a complex web of interrelated references in both the physical and digital worlds that they keep in their head, and that they struggle to map this structure into a single organizational form.

Since these studies were conducted seven to twenty years ago, we wanted to know if scientists still struggled with managing their information. We are particularly interested in how current tools support or hinder knowledge management and sharing in their daily tasks.

Data Analysis Tasks and Workflows

Data analysis and sharing is an essential part of scientific discovery [24]. The iterative and exploratory nature of data analysis [8] tends to produce numerous manually versioned scripts and output files [9]. Computational notebooks combine code, visualizations and text, bringing fragmented but related information into a single document [31]. Scientists are rapidly adopting this new medium to document and share exploratory data analyses [35]. Researchers have built tools for computational notebooks to support various activities such as informal and rapid version control [15], re-finding of past analysis choices [16], easy navigation [30] and managing messes [11].

However, building these tools within computational notebooks does not fully solve the problem: studies show that users of computational notebooks need to access diverse tools [43] and combine them with other tools and documents [31, 4]. A survey [43] of data scientists' collaborative practices reported 13 high-level categories of tools, including code editors, e.g. Visual Studio Code, computational notebooks, e.g. Jupyter Notebook, spreadsheets, data analysis tools, e.g. SPSS, document editing tools, e.g. LaTeX, and presentation software, e.g. Microsoft PowerPoint. Rule et al. [31] found that "individual notebooks rarely tell a story by themselves but are routinely combined with other notebooks, emails, slides, and ReadMe". Furthermore, their interviewees "even transferred outputs of the analysis to an entirely different medium (e.g. slides, word processing document) for easier review."

These studies suggest that scientists use a variety of tools in their data analysis and sharing process. We are interested in how they track the inter-connections between different documents and files in this process and why they choose to create multiple documents in the first place.

File Relationships

Previous research has explored file relationships to devise new ways to organize digital content, including using metadata [5], tagging [25] and activity-based computing [32, 42]. We focus here on provenance [12] and linking [38], which are particularly relevant to representing relationships among files.

Provenance

Jensen et al. [12] studied the provenance of files on desktops and showed that it helped reveal the work patterns of knowledge workers. Several tools are based on the idea of sourcing file relationships for version management [17, 14] and file retrieval [36, 6, 37]. For example, Connections [36] uses file relationships to enhance file searches; Leyline [6] integrates provenance into the query composition and result presentation of a search system; and TaskTrail [37] tracks file provenance through user events such as *copy-paste* and *save-as*, and represents it in a graph view to assist users in re-finding documents.

Karlson et al. [14] describe a system that tracks copy relationships among files, helping users to manage their versions. They introduce the 'versionset', a set of files that represent a user's concept of a document and found that file format is an important element of the user's conceptual model.

Linkdley et al. [17] developed the 'file biography' to encompass the provenance of a file and its propagation. Commercial user interfaces for version control systems such as Gitkraken[3] and SourceTree[4] represent file revisions in a time-based tree visualization[5], but these tools are designed primarily for coordinating work among programmers. Prior studies have found that scientists rarely use version control systems [15, 28] and that knowledge workers mostly use manual versioning [14, 10] because they find version control systems too complex.

We find two limitations to these systems: They require users to explicitly specify relationships among files, and the relationships are presented as non-interactive graphs. We are therefore interested in improving both automation to infer file dependencies and direct interaction to provide more user control.

[3] https://www.gitkraken.com/b

[4] https://www.sourcetreeapp.com/

[5] See a list of GUIs at https://git-scm.com/downloads/guis/.

Linking

Tabard et al. [38] take a different approach to support web navigation by automatically linking web pages together according to user actions. Software development tools such as webpack[6] generate a dependency graph by analyzing source code, but do not visualize nor let users control the resulting graph.

Our goal is to design a lightweight tool for knowledge workers to keep track of file dependencies and manage the various versions and formats of their files. Our target audience are users who manage information across files of various types and process them through tools such as scripts and compilers.

FORMATIVE STUDY: INTERVIEWS WITH SCIENTISTS

We conducted critical object interviews [19] to better understand how scientists, who can be considered "extreme" knowledge workers, manage complex research information.

Participants: We interviewed 23 scientists (7 women, 16 men): six professors, four researchers, four Ph.D. students, five research engineers, three post-docs, and one research associate. The research topics include neuroscience, mathematics, chemistry, physics, biology and computer science.

Procedure: All interviews were conducted in English by the second author and lasted 45-60 minutes each. We visited the participants in their labs and asked them to show us the tools, e.g. notebooks, that they use to keep track of their research information. Based on the artifacts they showed us, we asked them to describe a recent, memorable event, either positive or negative, related to using that artifact. We then asked about their work practices based on these events and artifacts.

Data Collection: All interviews were audio recorded and transcribed. We also took photos and hand-written notes.

Data Analysis: We analyzed the interviews using reflexive thematic analysis [3]. We generated codes and themes both inductively (bottom-up) and deductively (top-down), looking for breakdowns, workarounds and user innovations. Two coders (including the interviewer) conducted the analysis. They grouped codes into larger categories, discussed any disagreements and rechecked the interview transcripts to reach a shared understanding. They arrived at the final themes after two iterations.

RESULTS AND DISCUSSION

We used the concept of *information substrate* to conduct the thematic analysis. Beaudouin-Lafon [1] describes substrates as computational media that hold digital information, possibly created by other substrates, manage constraints and dependencies, react to changes in both the information and the substrate, and generate information consumable by other substrates. Maudet et al. [20] analyzed the constraints and rules that graphical designers manage in their heads and introduced digital tools that explicitly support these substrates. We follow a similar approach by identifying the types of substrates currently used by scientists in order to inform the design of more appropriate ones. We extend the original concept of substrate to cover both physical and digital artifacts.

[6] https://webpack.js.org/

Based on these concepts, the thematic analysis led to six primary themes related to: using multiple substrates; transferring information across substrates; keeping information "just in case"; finding and re-finding; iterating the produce-communicate cycle; and expressing file relationships.

Taking Advantage of the Characteristics of Substrates

All participants take advantage of the characteristics of different types of information substrates to manage their information. For example, they use paper as a thinking and reasoning tool because free-hand drawing enables rich representations, flexible layout and better memorization. This is consistent with prior studies on the advantages of paper [33, 26, 18, 34].

Most participants (20/23) use at least one blackboard or whiteboard in their office or in meeting rooms. These boards offer a large working space for free-hand sketching and writing, supporting fluid collocated collaboration [27, 40].

Five participants with experimental science training also keep a physical lab notebook. This notebook follows a chronological order and makes it easy to switch between structured and free-form information. Participants create links to keep related information in context by noting down file names or sticking print-outs on the relevant pages. Participants also have at least one personal notebook for "purely temporal information" (P15). These personal notebooks are less clean and structured than the "official" lab notebook. Participants transfer the important information to the lab notebook.

Similar to the analysis of biologists' work practice [39], these results suggest that the nature of information being distributed among different artifacts such as paper, whiteboard and lab notebook, is driven by their unique characteristics.

Transferring Information Across Substrates

From a first raw idea on paper to a well-written article, participants continuously structure their ideas over time as a project unfolds. We found that the process of transferring information from one substrate to another is a necessary cognitive process where participants review, rethink and re-evaluate their ideas.

P9 described how he worked with a colleague on an idea by writing on the blackboard; copying it onto paper in a cleaner version; going back and forth between blackboard and paper several times until it was mature enough to write it in LaTeX. He explained: "When I copy, I am not just copying. I try to understand it of course... We also fix errors when we transfer".

Seven participants reported taking whiteboard photos during meetings to persist information digitally. While taking a photo is quick, they often forget to do it (P9, 10) and find it cumbersome to transfer the photos to their laptop (P2), organize them (P2, 9, 21) and understand the messy content of the whiteboard after the fact (P22). As a result they prefer transcribing information onto a different substrate, such as their notebook.

Keeping Information "Just in case"

A quarter of the scientists (P1, 6, 9, 10, 14, 21) tend to hoard information (whiteboard photos, figures, versions of paper) "just in case" even though they rarely revisit it (P9, 10, 14), partially because they are not able to anticipate what will be

needed in the future. For example, P21 (computer scientist) said: "It is really insane that amount of photos we have. We clearly do not use all of them."

Hoarding information adds to the challenge of future re-finding. Participants find it difficult to determine the right version among their multiple devices: "Anyways, everything is mixed up (between home and work computers)." (P11). Participants use time stamps and modification events as primary cues to determine the right version. While prior research has focused on supporting document versions [14], we also found the need to support managing the versions and variants of graphs, particularly in scientists' data analyses. For example, P6 (bioinformatician) prints several versions of the same graphs as he iterates analyses and puts the latest version in a plastic folder to easily identify it.

Finding and Re-finding

All participants develop various strategies to support their future re-finding tasks.

Summarizing

Participants summarize information to avoid having to go back to the raw source. The summarized information is usually in another substrate that serves as a reference when re-finding. For example, P19 (experimental chemist) summarizes everything about a particular molecule in a physical album when she completes a series of experiments and has a clear result.

Linking related information

Five experimental scientists link related information together in their master lab notebook, like in [39]. If they need to link digital content, they simply note down the file names. For example, P15 (computational chemist) writes down all the information related to a plot in his paper lab notebook, including location of the data, scripts to generate that plot, location of the Jupyter notebook of that plot and, name and location of the image file.

However, these manual links with file names break when participants change the name or location of the file, as illustrated by P16 (experimental physicist): "I have done a lot of cleaning of my files so I don't know if I can find it."

Saving data in a single place

Eight data scientists use a project-based organization to save related information in a single folder, similar to [39]. But their tendency to hoard information and the messy results produced by exploratory programming [31] result in folder clutter. For example, P18 (machine learning researcher) saves the data, the analysis scripts and the output in a single folder. As he produces variations and versions of the scripts and corresponding outputs, files accumulate, hindering re-finding.

Several participants use computational notebooks, e.g. Jupyter notebooks, to keep track of their analysis process by saving scripts and their outputs in a single place. However, as Rules et al. [31], we found that these notebooks are routinely combined with other documents such as slides and ReadMe.

This suggests that the strategy of saving data in a single place might not be flexible enough for scientists' knowledge management. They also need a lightweight way of linking related but distributed content together.

Preserving importance

The transition from one substrate to another is also a process where participants assess the importance of information. They usually preserve important information in a new substrate in a more structured way and discard information they do not need anymore. For example, P10 (mathematician) first works on paper, then selects the ideas he thinks are worth preserving and transfers them onto his paper notebook.

This process also applies to data analysis. P14 (bioinformatician) performs exploratory data analysis in one Jupyter notebook, where he produces many graphs and statistics. Once he is happy with the results, he copy-pastes them into his main Jupyter notebook, where he collects all the important results.

Iterating the Produce-Communicate-Reproduce Cycle

Communicating results often involves manually cleaning, selecting and inputting content into another type of document or software tool. Note that the results are often in the form of plots and graphs. Participants use a range of media for sharing their results including Microsoft PowerPoint (P22, 6, 16) and Word (P22), Printout (P6, 13), LaTeX (P13), Google doc (P21), HTML (P15, 2), Markdown (P18) and Jupyter notebook (P14, 22). This supports the finding that data analysts transfer outputs of their analyses to an entirely different medium, e.g. slides or word processing document, for easier review [31].

Participants' choice of medium is related to the audience they target. For example, P13 prefers to give his supervisor a paper printout because she is more likely to read it and get back to him. P2 (bioinformatician) prefers to share a static HTML file instead of an interactive Jupyter notebook because "what we have done in the project is that every group has its own expertise. If they want to change the parameters, they will ask you to change it. They won't do it directly."

Some participants (P6, 22) complain that it is time-consuming to tediously copy-paste content into slides for easy sharing: "The problem is that you have to drag and drop into PowerPoint. So when you have 15 images, it is a bit time consuming." (P6)

Scientists often face re-finding challenges when they need to modify content after getting feedback. For example, P18 had to recreate a graph after discussing with coauthors but could not find the data for that graph: "I did so many experiments that I could have accidentally overwritten the file..."

After a modification, the new results need to be communicated again. Scientists need to manually re-execute the update pipeline by cleaning, selecting and putting content into another medium. P14 (bioinformatician) said: "There is obviously a problem of synchronisation. At some point, I generate the new version of a figure with slightly different parameters. I need to reload it in Overleaf. It happens often." Managing different file formats or variants adds more overhead, as P11 illustrated: "Including external files in LaTeX can get messy very very

Issue	Empirical Evidence
(HF) Hard to re-find related but distributed information	Related information is often distributed by necessity because scientists take advantage of the characteristics of different information artifacts (both physical and digital), resulting in difficult information re-finding.
(LT) Lack of tools to track inter-file relationships	Among various strategies to support future re-finding, linking is effective but limited because participants want to create different types of relationships among files and using file naming conventions is error-prone.
(MS) Manual synchronization	To share results, participants manually re-execute the pipelines that create the target documents.
(VM) Difficult version management	Participants' hoarding behavior and use of multiple devices adds to the challenges of re-finding and versioning. They have difficulty keeping track of versions of their documents, such as graphs.
(FV) Lack of support to manage file variants, e.g. format	Participants produce variants of a file for different purposes. Different variants of the same file clutter the folder, making it hard to manage.
(SD) Tedious creation of shared document	Participants complain that it is time-consuming and tedious to re-create shared document for review.

Table 1. Six issues observed in the formative study

quickly. What? I want to use jpeg; I am pretty sure this thing is jpeg. But the browser is suggesting that it is not a jpeg."

In summary, the produce-communicate-reproduce cycle requires scientists to successfully re-find the original content, reproduce the results with new settings while keeping the previous versions and manually synchronizing the updated changes to another medium to communicate these results again.

Expressing Inter-File Relationships

We identified three types of inter-file relationships that scientists want to express: description, dependency and coexistence.

Description

To help re-find and re-understand in the future, participants often use text files to add explanations to other files, e.g. tables, images, pictures and code. Participants use ReadMe files (P14), tables of contents in notebooks (P15) and code comments (P6). For example, P21 (computer scientist) writes plain text files for each image in a folder to help re-find them.

Re-finding code is particularly challenging because it requires re-understanding the code. P14 (bioinformatician) finds computational notebook useful but still writes additional ReadMe files for each Jupyter notebook to explain the analysis to his future self [31]. Other participants complain that out-of-order execution and unclear cell dependencies hinders the reproducibility of previous results. For example, P15 does not use a Jupyter notebook as a full-fledged lab notebook because he can accidentally re-run a code cell that overwrites the figure he wants to save. He still uses his paper notebook.

Dependency

Participants need to keep track of files that are generated by other files and manually keep them in sync. This often happens in produce-communicate-reproduce cycle when scientists need to manually modify and remix content across applications.

Coexistence

All eight data scientists put the data and analysis scripts (including Jupyter notebook) in sub-folders of the same folder to link the data with the corresponding analyses. P14 (bioinformatician) and his student created two separate but linked Jupyter notebooks because they use two languages (R and Python), that Jupyter does not support simultaneously.

Participants rely on file names to express inter-file relationships. However, links with file names are easy to break in case of renaming. File names are also used for other information such as experiment parameters, manual version suffix and file format, resulting in long, hard-to-understand file names.

Summary

This study shows that related information is often distributed *by necessity* because scientists take advantage of the characteristics of different information substrates. Their hoarding behavior complicates their versioning practices. They struggle to keep track, re-find and maintain consistency of this related but distributed information, particularly when collaborating asynchronously with students or colleagues.

FILEWEAVER

Based on the findings from the user study, we decided to focus on the file-related issues listed in Table 1. We designed *FileWeaver*, a prototype that augments traditional file management tools with automatic tracking and interactive visualization of file dependencies and histories. *FileWeaver* detects file dependencies and can update the dependents of a file when that file is changed. It also records file histories and can manage simultaneous versions and variants of a file.

FileWeaver User Interface

FileWeaver's interface has three interactive views (Fig. 1):

- A Folder View that displays all files in the current directory, similar to a typical file browser;
- A Graph View that displays files as nodes and dependencies as directed edges. For example, a file containing a dataset loaded by a script is connected to the file holding the script;
- A History View that displays the history of a file as an interactive tree, where each new *version* is connected to the previous one.

In all three views a menu with *FileWeaver* commands is accessible by right clicking a target or the background. The currently implemented commands are listed in Table 2.

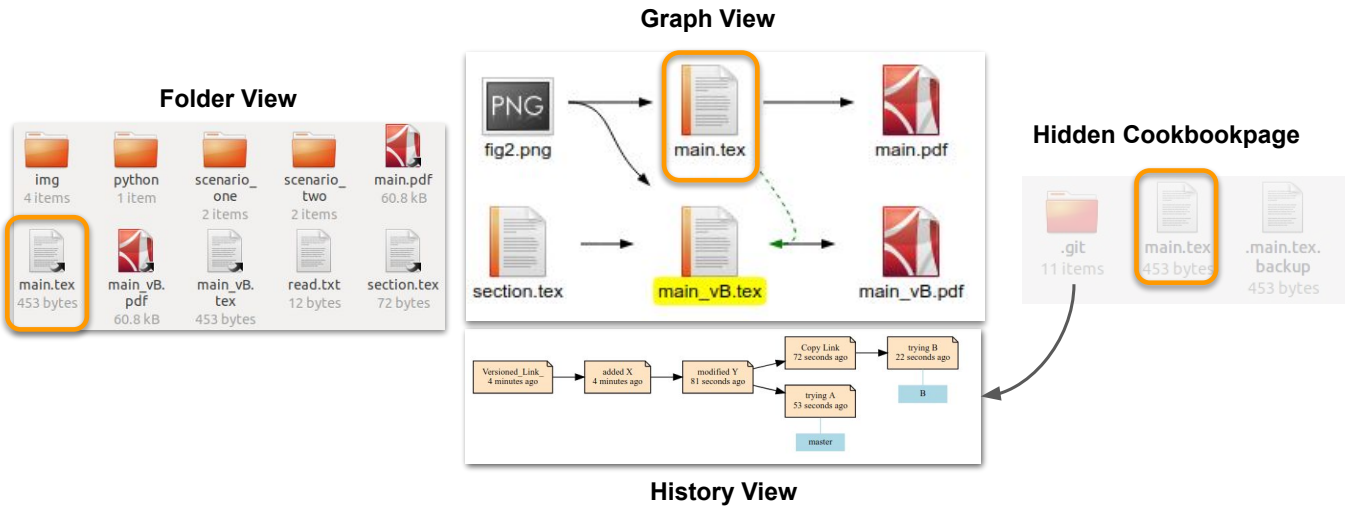

Figure 1. The *FileWeaver* User Interface. The Folder View (left) is a standard file browsing window. Files with arrows are symbolic links, meaning that they are managed by *FileWeaver*. The Graph View (center top) was displayed by selecting `main.tex` in the Folder View, and shows the dependency graph for that file. `main_vB.tex` is a copy of `main.tex` (green arrow); both files include `fig2.png` and `section.tex`, and they each produce a PDF file. The History View (center bottom) shows the history of versions of `main.tex`, including the branch that led to `main_vB.tex`. The Hidden Cookbookpage (right) holds the actual file and its version repository, together with bookkeeping information.

Folder View

FileWeaver uses the standard Folder view of the host operating system (Fig. 1 left), with an added sub-menu in the context menu of each file to invoke *FileWeaver* commands. The key command is *Add File*, which lets users add files to *FileWeaver* together with their dependencies. Files managed by *FileWeaver* have a little arrow on their icons to provide a visual cue.

File Dependencies and the Graph View

Each file added to *FileWeaver* is a node in a graph, and the dependencies between files are directed edges between the nodes. This graph is constructed and updated automatically. Whenever the user selects a file managed by *FileWeaver* in the Folder view, she can immediately see the dependency graph of that file in a separate Graph view. Most *FileWeaver* commands are available in both the Folder and Graph views.

Black edges represent regular, up-to-date dependencies. Other colors represent a different status: a green dashed edge indicates a *FileWeaver* copy (Fig. 1); a red edge indicates a stale dependency; a gray edge indicates a manual dependency, e.g. between a ReadMe file and a script; and a blue edge indicates the morph group of a file (Fig. 3).

Version Control and the History View

FileWeaver puts each file it manages under version control. When the user edits the file via the *Edit File and Update* command and saves it, *FileWeaver* asks her to enter a commit message and records a new version upon closing that file.

The *Show History* command opens a History view where each node represents a version, labeled with the first line of the commit message and the time it was created. Branches corresponds to the use of the *FileWeaver Copy* command. Using the context menu, the user can open any version of a file and compare two versions with color-highlighted differences.

Synchronization between Views

The Folder, Graph, and History views are synchronized and kept consistent. When the user selects a file in the Folder view, the Graph view updates to display the relevant graph. When *FileWeaver* detects that the dependencies of a file have changed, the graph is animated. The user can open a new Folder view at the location of any selected file in the Graph view, and a History view from either the Folder or Graph view.

This makes navigating the file system flexible: the Folder view provides traditional navigation of the file hierarchy, the Graph view displays the dependencies of a file, irrespective of their location in the hierarchy, and the History view lets users explore the evolution of a given file over time.

Use Cases

The combination of synchronized views, automatic dependency tracking and updating, and automatic versioning opens up new possibilities for managing files. We introduce the main features that we have implemented in *FileWeaver* through a scenario inspired by the thematic analysis, starring Jane, a data scientist. For each segment, we describe the scenario without *FileWeaver* and reference (in italics) the issues found in the thematic analysis. Then we describe the scenario with *FileWeaver*, showing how it addresses the issues.

Detecting and Maintaining Dependencies

FileWeaver automatically tracks the dependencies between files and displays them in the Graph view. When a new file is added to *FileWeaver* (either manually by the user or automatically by *FileWeaver* itself), *FileWeaver* detects its dependencies and adds them to the graph. When a user edits a file via the *Edit and Update File* command, e.g. by double-clicking it in the Graph view, *FileWeaver* updates all the dependent files and displays the result file. For example, if the user updates a LaTeX source file, *FileWeaver* opens the resulting PDF file.

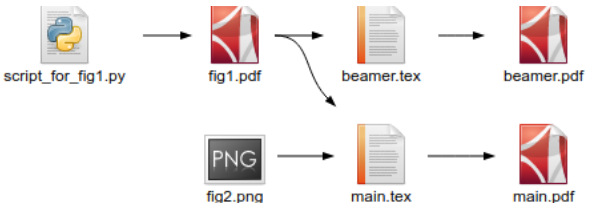

Figure 2. Graph view for the scenario (see text)

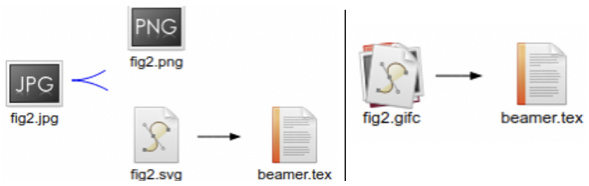

Figure 3. Polymorphic file in expanded (left) and collapsed (right) form.

Scenario without *FileWeaver*: Jane produces a figure via a Python script and wants to include it in a LaTeX document. She moves the figure to the document's directory (*Save in a single place*). She also wants to create slides in Beamer (*Communicate*) with that figure, so she also copies the figure to her Beamer directory (*Save in a single place*). Every time she needs to change the figure (*Produce-communicate-reproduce*), she has to run her Python script, copy-paste the figure twice to replace the old ones, and compile the LaTeX and Beamer files. To send a standalone version of her project, Jane has to go through various directories to find all the relevant files and the right versions of the scripts and send them over via email.

Scenario with *FileWeaver*: Jane adds the script to *FileWeaver*. *FileWeaver* runs the script and two nodes (for the script and figure) appear in the Graph view. She then adds the LaTeX file that includes the figure. *FileWeaver* compiles it and two new nodes (LaTex file, output PDF file) appear in the Graph view, with edges from the figure to the LaTeX file, and from the LaTeX file to the PDF file.

When she edits the script with *Edit and Update File*, *FileWeaver* updates all the dependent files by running the script and recompiling the LaTeX file, and opens the PDF file, with the up-to-date figure. When creating the Beamer file, Jane uses the Graph view to find the figure and opens its enclosing folder using the *Locate File* command. She can thus decide on the best file storage strategy. She then adds the Beamer file to *FileWeaver* (Fig. 2), so that any future change to the script will also recompile the slides.

Jane writes a ReadMe file and manually connects it to the script. The corresponding edge in the Graph View appears in gray. To send the project, she selects any file in the Folder or Graph View and issues the *Make Archive* command. This creates a `.zip` archive of all the files connected to that file.

Managing Variants with Polymorphic Files

FileWeaver introduces a new concept called polymorphic files—or *morphs* for short. A morph replaces a collection of files, such as different formats for the same image (Fig. 3), by a single file. We have implemented morphs for image files. To create a morph, the user selects a set of files, e.g. four pictures `pic.pdf`, `pic.png`, `pic.jpg`, `pic.svg` and issues the *Morph Files* command, which creates a single file for the morph with a `.gifc` extension (for **g**eneric **i**mage **f**ormat **c**ontainer). The user can import the morph into any other file, and *FileWeaver* will determine which variant is appropriate.

Scenario without *FileWeaver*: Jane creates a very large plot and saves it as an SVG file. To facilitate importing and sharing this figure, she converts the file into PDF and JPG. She now has to maintain three files by hand whenever the source of the plot changes (*Dependency*).

Scenario with *FileWeaver*: Jane creates the three image files as above, and morphs them into a single file. Whenever she includes the morph in a new file, it uses the default image format for this file type. To use another format, she edits the edge attribute with the desired extension, which will from now on select the matching instance of the morph.

Managing History with Versions

Whenever a user edits a file via the *Edit and Update File* command, *FileWeaver* prompts for a descriptive message and saves the file as a new version. The user can explore the file versions in the History view, where each item displays the subject and date of the message, and compare any two versions using a GUI diff tool (Fig. 4).

Users who are not familiar with version control systems typically create their own versioning by making copies of files and editing them in parallel. *FileWeaver* provides the *Copy File with Dependencies* command, which lets the user work simultaneously on two versions of the files. These dependencies are shown in the Graph View as dashed green edges (Fig. 1).

Scenario without *FileWeaver*: Jane starts implementing a new feature. She makes several attempts and saves them under nondescript names (*Keeping information "just in case"*). A few weeks later, she returns to it but cannot figure out which file she was working on. She may even have overwritten it.

Scenario with *FileWeaver*: Jane creates a copy of her script with the *Copy File with Dependencies* command, so she can work on it independently from the original script. The copy shows up in the Graph view, linked to the original file with

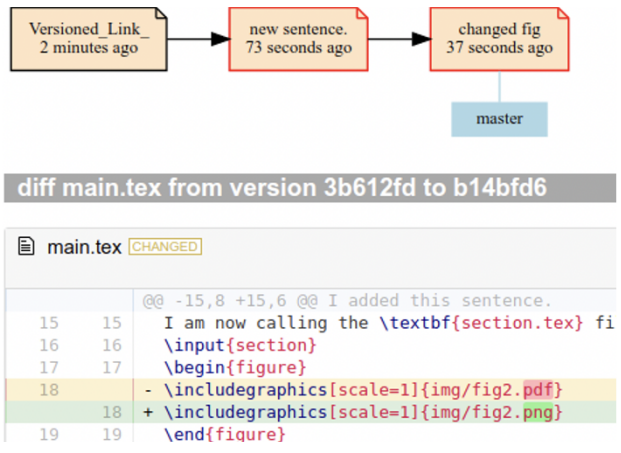

Figure 4. History view showing the differences between two versions.

a green edge. As she works on the new feature, *FileWeaver* stores the successive versions. When she comes back to it a few weeks later, she uses the History View to display the differences between her last edits and remember her progress. When the feature is ready, she merges the copy into the original. If she gives up instead and deletes the copy, she can always get back to it later via the History view of the original file, which shows the branch corresponding to the copy.

How FileWeaver Works

We now describe how *FileWeaver* supports the features described above.

Graph Attributes

Nodes and edges of the graph have attributes that are stored in a separate database. Some attributes are maintained automatically, e.g. *edge update time*, which tracks when the dependency between two files was last brought up to date. *FileWeaver* compares it to the file's UNIX `mtime` attribute to decide whether to update the dependency. Each node also stores the version ID of the corresponding file, and each edge the version ID of the source side of the edge. If two files (A and B) depend on different versions of a third file (C), the edge (say, from C to A) representing the dependency that is not up to date is shown in red. This tells the user that A should be updated. If instead A and B were produced from C as part of the produce-communicate-reproduce cycle, it tells the user that the version of C that was used to produce A is accessible from the History View, whereas it would normally have been overwritten by the version that produced B.

Other attributes are under user control. For example, The user can set *node update* to false to specify that this node should not be automatically updated, e.g. because it is too computationally expensive to run the update. The user can also change the *recipes* for a given file (see next).

Recipes: Tracking and Updating Dependencies

FileWeaver uses file-dependent scripts called *recipes*, stored as node attributes, to track and maintain dependencies. These recipes specify how each file is processed (*update*), displayed to the user (*interact*) or its dependencies tracked (*trace*), allowing a high level of task automation and the ability to always keep the Graph view up to date

The *update* recipe processes the file and produces its output if there is one. For example, the recipe for a LaTeX file can simply be `latexmk $filename`. The *update* recipe is run whenever a file needs to be updated. When a user triggers a *FileWeaver* command on a given file, *FileWeaver* checks that the dependent files are up to date, in a topological sorting order, and launches the *update* recipe on the target file if an update is indeed needed.

The *trace* recipe is optional and should produce the same file accesses as *update*, only faster. If unspecified, the *update* recipe is used instead. For example, the *trace* recipe for a LaTeX file can be simply `pdflatex $filename`. *FileWeaver* traces the file accesses resulting from running the *trace* recipe when the file dependency list needs updating, e.g. when a file is edited or newly added. *Trace* recipes are motivated by files with long compilation procedures, such as LaTeX files: while the *update* call to `latexmk` will usually result in at least three calls to `pdflatex`, the *trace* recipe only needs a single one. *Update* and *trace* recipes should include clean-up commands, if needed, to remove temporary files and avoid cluttering.

The *interact* recipe opens an editor or viewer for the file. For a LaTeX file, it calls the user's preferred LaTeX editor, e.g. `texmaker -n $filename`. *FileWeaver* runs the *interact* recipe when the user edits the file through *FileWeaver*, or when *FileWeaver* wishes to display a file to the user, e.g. to show the result of an update. When the user closes the editor (which exits the *interact* recipe), changes are automatically versioned. Note that the user can always edit a file outside of *FileWeaver*; it will then be recognized as out of date on the next call to a *FileWeaver* command.

When a file is added to *FileWeaver*, the recipes are initialized according to the file's type. Each file type has a default set of recipes stored in a `.rcp` file. *FileWeaver* currently features recipes for 9 popular file types: `.tex`, `.py`, `.pdf`, `.png`, `.jpg`, `.jpeg`, `.csv`, `.svg`, `.txt`. The default recipes for `.tex` files is given in the Appendix. Users can edit `.rcp` files, share them, and specify custom recipes for any given file.

We acknowledge that writing recipes may require some computing knowledge. We expect to provide more recipe to cover more file types, and a simple editor to facilitate creating, editing and sharing them.

Links and the Cookbook

The *FileWeaver* back-end runs in the background and strives to be as transparent as possible to the user. It uses a hidden folder called the *cookbook* to store its files.

Each file managed by *FileWeaver* is attributed a folder, called a *cookbook page*, in the cookbook. When *FileWeaver* starts tracking a file, that file is moved to a new cookbook page, and a symbolic (soft) link is created at the original file location, pointing to the file in its cookbook page. This leaves the user's folder virtually unchanged, except for the fact that the user now sees a symbolic link rather than a standard file. We use soft links rather than hard links because the latter do not span filesystems (including partitions). Also, most file browsers display symbolic links with an arrow on top of the file icon, giving the user a clue that the file is managed by *FileWeaver*.

A cookbook page also holds the version control repository for that particular file (Fig. 1 right). Cookbook pages are named after the device and i-node number of the file that they store, making for a cheap and memoryless, hashtable-like one-to-one mapping between files and cookbook pages[7].

Tracking File History Through Version Control

Whenever a file is edited via the *Edit and Update File* command, *FileWeaver* prompts the user for a message and automatically commits the file to the version control repository

[7] Some text editors use temporary files that are renamed into the original file after saving, therefore changing the i-node number of the file. *FileWeaver* creates an additional, hidden hard link to the file when it is added to the cookbook page to prevent the i-node from changing. This hard link can also serve as a backup access to the file.

located in the file's cookbook page. While *FileWeaver* could automatically detect file changes and perform actions, we avoided this behavior to give users a better sense of control. The *FileWeaver Copy* command creates a new cookbook page for the copy but shares the same repository as the original file so that the user can merge the two files with the *Merge Files* command. All edges to parent nodes and attributes are copied from the original file, preserving custom settings.

Polymorphic files

When the user select files and invokes the *Morph Files* command, *FileWeaver* adds these files to its cookbook as usual, and assigns them to a *morph group*. Instead of creating a soft link for each file, it creates a single soft link for the morph with a `.gifc` extension (**g**eneric **i**mage **f**ormat **c**ontainer). When a file calls a `.gifc` extension, *FileWeaver* goes through the graph and looks if there is an edge between that file and a member of the morph group. If so it redirects the soft link with the `.gifc` extension to that morph group member; otherwise it goes through the default image formats associated with that file, e.g. pdf/png/jpg for a LaTeX file, and creates an edge with the corresponding morph group member. Whenever a file is used by *FileWeaver*, e.g. as part of an update, *FileWeaver* checks if that file uses a morph. If so, it makes sure that the soft link is pointing to the right member of the morph group. Since some systems, e.g. LaTeX, use the file extension of included files, *FileWeaver* creates a second, temporary soft link with the right extension, and runs a stream editor to search and replace the morph file name with the right extension. When the update is complete, the original file content is restored and the second soft link is removed.

Implementation

FileWeaver runs on Linux Ubuntu 18.04 LTS. The back-end is written in just under 5000 lines of Python with calls to standard UNIX tools such as `bash` and `sed`[8]. File dependencies are detected by running `Tracefile`[9] on the *trace* recipe to track file accesses. The graph database is managed via the `graph-tool` library[10]. The back-end currently runs only on Linux due to the use of `Tracefile` and `strace`, but we are considering alternatives for Mac OS.

The Folder view is the GNOME Nautilus file manager[11] (Fig. 1 left). We use `nautilus-python`[12] to write extensions to Nautilus for running the backend and adding the menu of *FileWeaver* commands to the standard Nautilus file menu.

Both the Graph and History views (Fig. 1 center) are implemented with NWJS[13], a platform to create web-based desktop applications. The Graph View sends commands to the back-end through a simple pipe, and receives updates to the graph through a shared file. It uses `dot`[14] for the layout of the graph. The version control system is Git[15]. Although not designed to deal with binary files such as figures, it can handle a moderate amount of them without problem. We plan to explore extensions to Git that deal with large or numerous binaries[16]. The History View directly calls Git commands for the different version management commands. It uses `git2dot`[17] to create the version tree and `diff2html`[18] to display version diffs. The *FileWeaver Copy* command uses the `git-worktree` feature to check out several branches at a time.

DISCUSSION AND EVALUATION

Table 2 lists the most important features implemented in *FileWeaver* and the issues identified in the formative study that they address (see Table 1). We complement this analysis with a more general qualitative evaluation of *FileWeaver* based on Green et al.'s cognitive dimensions [7]. We leave a formal, longitudinal user study to future work.

Visibility and Hidden Dependencies

Visibility is the ability to view components easily, while hidden dependencies reflects whether important links between entities are visible or not. *FileWeaver* achieves high visibility and low hidden dependencies by making dependencies among files explicit and visible in the Graph view. Dependencies are represented by edges that can be manipulated, e.g. change the update rule. Automatically running recipes also reveal dependencies as files are edited, unlike tools such as, e.g. Makefiles, that require manual editing to describe the dependencies.

In typical folder-based views, all non-hidden files are visible, which may clutter the folder. *FileWeaver*'s polymorphic files let users view the multiple variants of a file as a single entity, making their relationship more explicit and reducing clutter.

Viscosity

Viscosity refers to resistance to change. *FileWeaver* has low viscosity because it automatically propagate the user's changes to a file to its dependents, achieving consistency between files without explicit user action.

FileWeaver also automatically stores successive versions of a file so that users can examine and revert to previous versions using the History view. It supports exploration of alternatives by making it easy to create parallel copies and merge them.

FileWeaver prompts the user for a message when a file is edited via *Edit and Update File*, which has high viscosity. Although encouraging users to document their changes can help them re-find the right version in the future, this requires extra effort. We plan on simplifying this process, e.g. by suggesting auto-generated commit messages.

Error-proneness

FileWeaver reduces errors by automating the processes of dependency update and versioning. The former reduces the

[8] https://www.gnu.org/software/

[9] https://github.com/ole-tange/tangetools/blob/master/tracefile/tracefile.pod, which is a wrapper around Linux's standard `strace` (https://linux.die.net/man/1/strace).

[10] https://graph-tool.skewed.de/

[11] https://github.com/GNOME/nautilus

[12] https://github.com/GNOME/nautilus-python

[13] https://nwjs.io/

[14] https://www.graphviz.org

[15] https://git-scm.com/

[16] such as git-lfs, git bup, git-annex

[17] https://github.com/jlinoff/git2dot

[18] https://diff2html.xyz

Command Name	*FileWeaver* Action	What users see	Issue Addressed
Add File as Link	Start managing the file and add all dependent files	New nodes and edges appear in the Graph view. The files become links in the Folder view	MS, HF, LT
Copy File with Dependencies	Copy a file with *FileWeaver* with all its user-defined *FileWeaver* attributes	A node is added to the Graph view, pointing from the original node by a green arrow.	LT, VM
Make Standalone Archive (Runnable and Flat Mode)	Export all dependencies as a `.zip` archive with/without parent directories	*FileWeaver* creates the `.zip` archive in the current Folder view location	SD, LT
Tag File / Group	Change the tag (label) attribute of a file (or group of files)	Labels in the Graph view set to the tag.	HF
Edit File and Update	Run the file's *interaction* script; commit changes upon completion. Update dependents in topological order; run *interaction* script on last file	An editor pops up; on closing it, a text window prompts for a commit message. The Graph view is updated and the new output file is displayed	MS, VM
Locate File	Change the Folder view to the directory containing the file	The file is highlighted in the new Folder view	HF
Connect File	Add an edge between two nodes in the Graph view	The edge appears in light grey, different from other edges in the Graph view.	LT
Disconnect File	Remove a permanent edge between two nodes in the Graph view	The edge disappears from the Graph view	LT
Show File History	Populate the History view with the Git history tree	Each node in the history represents a commit. *FileWeaver* copies appear as branches. Each node displays commit date and subject	VM
Morph Files	see § Polymorphic Files	Morphed files are linked together with blue arrows in the Graph view	FV
Compare File Versions	Call Git difftool	A GUI diff tool appears in the History view to compare files	VM

Table 2. How different features of *FileWeaver* address issues observed in the formative study

risk of running the wrong command to update files, and the latter makes it possible to get back to a former version in case of an error. In future work we plan to use version control also for the graph itself, which would make it possible to easily recover deleted files and dependencies.

Secondary Notation

Secondary notation refers to the ability to carry additional information. *FileWeaver* lets users tag files to rename the node labels while keeping the underlying files with their original name. Users can also edit the *recipe* and *interact* scripts of individual files to tailor them to their needs. In future work we will make it easier to define file types and edit default recipes, as they are not currently readily accessible to novice users.

Role Expressiveness

Role expressiveness refers to how obvious the role of each component is. Each view in *FileWeaver* has a specific role: Folder view for regular file management, Graph view for managing dependencies, and History view for versioning. Role expressiveness could be further improved by decoupling components and giving users more flexible ways to combine them.

Premature Commitment

Premature commitment refers to constraints on the order to complete tasks. *FileWeaver* limits premature commitment by letting users add files to the system whenever they want. File names, contents and locations can also be modified at any time. Asking for a commit message when saving a file is a form of premature commitment, and should be made more flexible.

CONCLUSION AND FUTURE WORK

Knowledge management and sharing involves various specialized but isolated software tools, tied together by the files that these tools use and produce. We interviewed 23 scientists and showed that the information they manage is often distributed, because they take advantage of the characteristics of different information artifacts and rely on specialized tools. Although scientists develop strategies to cope with re-finding information, they still struggle to manage versions, maintain consistency and keep track of related distributed information.

We introduced *FileWeaver*, a prototype file management system where the traditional folder view is augmented by an interactive Graph View that displays dependencies among files and a History View that lets users interact with the different versions of a file, supporting navigation and re-finding tasks. *FileWeaver* also features polymorphic files, which groups the different variants of a file into a single, generic format. Together, these features create a new information substrate that explicitly represents the web of file dependencies and relationships that users typically have to manage in their heads.

FileWeaver helps automate the workflows of knowledge workers who use files to manage information and tools such as scripts and compilers to process them. It weaves files into a graph of dependencies, providing users with an interactive visualization of these dependencies. It weaves the history of each file into a version tree, providing users with an interactive temporal view of the file. Finally it weaves existing tools that are part of the user's workflow together to increase automation.

We plan to evaluate *FileWeaver* in a realistic setting to gather feedback and improve the interface. We also need to assess its scalability on larger sets of files. Future improvements include making recipes editable by novice users, including through automatic capture of commands; simplifying the commit process, e.g. by suggesting commit messages; and detecting dependencies in files such as zip archives and Word documents. Finally, *FileWeaver* has great potential for collaboration, since the file contents and its relationships are under version control and can therefore be pushed to a remote server for sharing.

ACKNOWLEDGMENTS

We thank Injung Lee for her help analyzing data for the interview study. This work was partially supported by European Research Council (ERC) grant n° 695464 ONE: Unified Principles of Interaction.

REFERENCES

[1] Michel Beaudouin-Lafon. 2017. Towards Unified Principles of Interaction. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter (CHItaly '17)*. Association for Computing Machinery, New York, NY, USA, Article 1, 2 pages. DOI: http://dx.doi.org/10.1145/3125571.3125602

[2] Olha Bondarenko and Ruud Janssen. 2005. Documents at Hand: Learning from Paper to Improve Digital Technologies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. Association for Computing Machinery, New York, NY, USA, 121–130. DOI: http://dx.doi.org/10.1145/1054972.1054990

[3] Virginia Braun and Victoria Clarke. 2019. Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (2019), 589–597. DOI: http://dx.doi.org/10.1080/2159676X.2019.1628806

[4] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. DOI: http://dx.doi.org/10.1145/3313831.3376729

[5] Paul Dourish, W. Keith Edwards, Anthony LaMarca, and Michael Salisbury. 1999. Presto: An Experimental Architecture for Fluid Interactive Document Spaces. *ACM Trans. Comput.-Hum. Interact.* 6, 2 (June 1999), 133–161. DOI: http://dx.doi.org/10.1145/319091.319099

[6] Soroush Ghorashi and Carlos Jensen. 2012. Leyline: Provenance-Based Search Using a Graphical Sketchpad. In *Proceedings of the Symposium on Human-Computer Interaction and Information Retrieval (HCIR '12)*. Association for Computing Machinery, New York, NY, USA, Article Article 2, 10 pages. DOI: http://dx.doi.org/10.1145/2391224.2391226

[7] T. R. G. Green. 1990. Cognitive Dimensions of Notations. In *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*. Cambridge University Press, USA, 443–460.

[8] Philip Jia Guo. 2012. *Software tools to facilitate research programming*. Ph.D. Dissertation. Stanford University Stanford, CA.

[9] Philip J. Guo and Margo Seltzer. 2012. BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure. In *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance (TaPP'12)*. USENIX Association, Berkeley, CA, USA, 7. http://dl.acm.org/citation.cfm?id=2342875.2342882

[10] Han L. Han, Miguel A. Renom, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2020. Textlets: Supporting Constraints and Consistency in Text Documents. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. DOI:http://dx.doi.org/10.1145/3313831.3376804

[11] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 270, 12 pages. DOI:http://dx.doi.org/10.1145/3290605.3300500

[12] Carlos Jensen, Heather Lonsdale, Eleanor Wynn, Jill Cao, Michael Slater, and Thomas G. Dietterich. 2010. The Life and Times of Files and Information: A Study of Desktop Provenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. Association for Computing Machinery, New York, NY, USA, 767–776. DOI: http://dx.doi.org/10.1145/1753326.1753439

[13] David R Karger and William Jones. 2006. Data unification in personal information management. *Commun. ACM* 49, 1 (2006), 77–82.

[14] Amy K. Karlson, Greg Smith, and Bongshin Lee. 2011. Which Version is This? Improving the Desktop Experience within a Copy-Aware Computing Ecosystem. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2669–2678. DOI: http://dx.doi.org/10.1145/1978942.1979334

[15] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1265–1276. DOI: http://dx.doi.org/10.1145/3025453.3025626

[16] Mary Beth Kery, Bonnie E. John, Patrick O'Flaherty, Amber Horvath, and Brad A. Myers. 2019. Towards Effective Foraging by Data Scientists to Find Past Analysis Choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 92, 13 pages. DOI: http://dx.doi.org/10.1145/3290605.3300322

[17] Siân E. Lindley, Gavin Smyth, Robert Corish, Anastasia Loukianov, Michael Golembewski, Ewa A. Luger, and Abigail Sellen. 2018. Exploring New Metaphors for a Networked World through the File Biography. In

Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). Association for Computing Machinery, New York, NY, USA, Article Paper 118, 12 pages. DOI: http://dx.doi.org/10.1145/3173574.3173692

[18] Wendy E. MacKay. 1999. Is Paper Safer? The Role of Paper Flight Strips in Air Traffic Control. *ACM Trans. Comput.-Hum. Interact.* 6, 4 (Dec. 1999), 311–340. DOI: http://dx.doi.org/10.1145/331490.331491

[19] Wendy E Mackay. 2002. Using video to support interaction design. *DVD Tutorial, CHI* 2, 5 (2002).

[20] Nolwenn Maudet, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2017. Beyond Grids: Interactive Graphical Substrates to Structure Digital Layout. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 5053–5064. DOI: http://dx.doi.org/10.1145/3025453.3025718

[21] Kiran-Kumar Muniswamy-Reddy, David A Holland, Uri Braun, and Margo I Seltzer. 2006. Provenance-aware storage systems.. In *Usenix annual technical conference, general track*. USENIX Association, Berkeley, CA, USA, 43–56.

[22] Leonardo Murta, Vanessa Braganholo, Fernando Chirigati, David Koop, and Juliana Freire. 2014. noWorkflow: capturing and analyzing provenance of scripts. In *International Provenance and Annotation Workshop*. Springer, 71–83.

[23] Gerard Oleksik, Hans-Christian Jetter, Jens Gerken, Natasa Milic-Frayling, and Rachel Jones. 2013. Towards an Information Architecture for Flexible Reuse of Digital Media. In *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia (MUM '13)*. Association for Computing Machinery, New York, NY, USA, Article Article 12, 10 pages. DOI: http://dx.doi.org/10.1145/2541831.2541866

[24] Gerard Oleksik, Natasa Milic-Frayling, and Rachel Jones. 2012. Beyond Data Sharing: Artifact Ecology of a Collaborative Nanophotonics Research Centre. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. Association for Computing Machinery, New York, NY, USA, 1165–1174. DOI: http://dx.doi.org/10.1145/2145204.2145376

[25] Gerard Oleksik, Max L. Wilson, Craig Tashman, Eduarda Mendes Rodrigues, Gabriella Kazai, Gavin Smyth, Natasa Milic-Frayling, and Rachel Jones. 2009. Lightweight Tagging Expands Information and Activity Management Practices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Association for Computing Machinery, New York, NY, USA, 279–288. DOI: http://dx.doi.org/10.1145/1518701.1518746

[26] Kenton O'Hara and Abigail Sellen. 1997. A Comparison of Reading Paper and On-Line Documents. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. Association for Computing Machinery, New York, NY, USA, 335–342. DOI: http://dx.doi.org/10.1145/258549.258787

[27] Elin Rønby Pedersen, Kim McCall, Thomas P. Moran, and Frank G. Halasz. 1993. Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. Association for Computing Machinery, New York, NY, USA, 391–398. DOI: http://dx.doi.org/10.1145/169059.169309

[28] Santiago Perez De Rosso and Daniel Jackson. 2013. What's Wrong with Git? A Conceptual Design Analysis. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2013)*. Association for Computing Machinery, New York, NY, USA, 37–52. DOI: http://dx.doi.org/10.1145/2509578.2509584

[29] Pamela Ravasio, Sissel Guttormsen Schär, and Helmut Krueger. 2004. In Pursuit of Desktop Evolution: User Problems and Practices with Modern Desktop Systems. *ACM Trans. Comput.-Hum. Interact.* 11, 2 (June 2004), 156–180. DOI: http://dx.doi.org/10.1145/1005361.1005363

[30] Adam Rule, Ian Drosos, Aurélien Tabard, and James D. Hollan. 2018a. Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article Article 150 (Nov. 2018), 12 pages. DOI: http://dx.doi.org/10.1145/3274419

[31] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018b. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, Article Paper 32, 12 pages. DOI: http://dx.doi.org/10.1145/3173574.3173606

[32] Brandon Salmon, Steven W. Schlosser, Lorrie Faith Cranor, and Gregory R. Ganger. 2009. Perspective: Semantic Data Management for the Home. In *Proccedings of the 7th Conference on File and Storage Technologies (FAST '09)*. USENIX Association, USA, 167–182.

[33] Abigail Sellen and Richard Harper. 1997. Paper as an Analytic Resource for the Design of New Technologies. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '97)*. Association for Computing Machinery, New York, NY, USA, 319–326. DOI: http://dx.doi.org/10.1145/258549.258780

[34] Abigail J. Sellen and Richard H.R. Harper. 2003. *The Myth of the Paperless Office*. MIT Press, Cambridge, MA, USA.

[35] Helen Shen. 2014. Interactive notebooks: Sharing the code. *Nature* 515, 7525 (2014), 151–152.

[36] Craig AN Soules and Gregory R Ganger. 2005. Connections: using context to enhance file search. In *Proceedings of the twentieth ACM symposium on Operating systems principles*. Association for Computing Machinery, New York, NY, USA, 119–132.

[37] S. Stumpf, E. Fitzhenry, and T. G. Dietterich. 2007. The use of provenance in information retrieval. In *Workshop on Principles of Provenance (PROPR)*. 2. https://openaccess.city.ac.uk/id/eprint/224/

[38] Aurélien Tabard, Wendy Mackay, Nicolas Roussel, and Catherine Letondal. 2007. PageLinker: Integrating Contextual Bookmarks within a Browser. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. Association for Computing Machinery, New York, NY, USA, 337–346. DOI:http://dx.doi.org/10.1145/1240624.1240680

[39] Aurélien Tabard, Wendy E. Mackay, and Evelyn Eastmond. 2008. From Individual to Collaborative: The Evolution of Prism, a Hybrid Laboratory Notebook. In *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work (CSCW '08)*. Association for Computing Machinery, New York, NY, USA, 569–578. DOI: http://dx.doi.org/10.1145/1460563.1460653

[40] Anthony Tang, Joel Lanir, Saul Greenberg, and Sidney Fels. 2009. Supporting Transitions in Work: Informing Large Display Application Design by Understanding Whiteboard Use. In *Proceedings of the ACM 2009 International Conference on Supporting Group Work (GROUP '09)*. Association for Computing Machinery, New York, NY, USA, 149–158. DOI: http://dx.doi.org/10.1145/1531674.1531697

[41] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. 2004. The Perfect Search Engine is Not Enough: A Study of Orienteering Behavior in Directed Search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. Association for Computing Machinery, New York, NY, USA, 415–422. DOI: http://dx.doi.org/10.1145/985692.985745

[42] Stephen Voida and Elizabeth D. Mynatt. 2009. It Feels Better than Filing: Everyday Work Experiences in an Activity-Based Computing System. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Association for Computing Machinery, New York, NY, USA, 259–268. DOI:http://dx.doi.org/10.1145/1518701.1518744

[43] Amy X. Zhang, Michael Muller, and Dakuo Wang. 2020. How Do Data Science Workers Collaborate? Roles, Workflows, and Tools. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW1, Article 022 (May 2020), 23 pages. DOI:http://dx.doi.org/10.1145/3392826

APPENDIX

A `.rcp` file stores the three scripts used by *FileWeaver* by delimiting them with single-line markers: `=DEFAULT-UPDATE`, `=DEFAULT-TRACE`, `=DEFAULT-INTERACT`. It also stores the preferred image formats between the `=DEFAULT-FORMAT-IMG` marker, in case a *morph* is used. The name of the file is that of the file suffix it is associated with, e.g. `tex.rcp` for LaTeX files. The `.rcp` file for LaTeX files is as follows:

```
=DEFAULT-UPDATE
#!/bin/bash
if cat $1 | grep -q \begin{document};
then
    path=$(dirname "$1")
    cd $path
    latexmk -pdf -interaction=nonstopmode \
        -bibtex-cond -silent -deps-out=.deps.txt $1
    if head -1 $1 | grep -q {beamer};
    then
        latexmk -c $1
        rm *.nav
        rm *.snm
    else
        latexmk -c $1
    fi
    exit 1
else
    exit 1
fi
!=DEFAULT-UPDATE
=DEFAULT-TRACE
#!/bin/bash
if cat $1 | grep -q \begin{document};
then
    path=$(dirname "$1")
    cd $path
    pdflatex -output-format pdf -interaction \
        nonstopmode $1
    if head -1 $1 | grep -q {beamer};
    then
        latexmk -c $1
        rm *.nav
        rm *.snm
    else
        latexmk -c $1
    fi
    exit 1
else
    exit 1
fi
!=DEFAULT-TRACE
=DEFAULT-INTERACT
texmaker -n $1
!=DEFAULT-INTERACT
=DEFAULT-FORMAT-IMG
pdf/eps/png
!=DEFAULT-FORMAT-IMG
=END
```