

Exploration and Explanation in Computational Notebooks

Adam Rule

Design Lab, UC San Diego
La Jolla, CA
acrule@ucsd.edu

Aurélien Tabard

Université de Lyon, CNRS,
LIRIS UMR5205, Lyon, France
aurelien.tabard@univ-lyon1.fr

James D. Hollan

Design Lab, UC San Diego
La Jolla, CA
hollan@ucsd.edu

ABSTRACT

Computational notebooks combine code, visualizations, and text in a single document. Researchers, data analysts, and even journalists are rapidly adopting this new medium. We present three studies of how they are using notebooks to document and share exploratory data analyses. In the first, we analyzed over 1 million computational notebooks on GitHub, finding that one in four had no explanatory text but consisted entirely of visualizations or code. In a second study, we examined over 200 academic computational notebooks, finding that although the vast majority described methods, only a minority discussed reasoning or results. In a third study, we interviewed 15 academic data analysts, finding that most considered computational notebooks personal, exploratory, and messy. Importantly, they typically used other media to share analyses. *These studies demonstrate a tension between exploration and explanation in constructing and sharing computational notebooks.* We conclude with opportunities to encourage explanation in computational media without hindering exploration.

Author Keywords

Computational notebook; Jupyter Notebook; data science; data analysis; narrative;

ACM Classification Keywords

H.5.3. Information interfaces and presentation: Group and Organization Interfaces – Collaborative computing

INTRODUCTION

Data analysis is an iterative and exploratory process of extracting insights from data [11, 17]. Insights are sensitive to the methods used to produce them; small changes in how data are collected, cleaned, or processed can lead to vastly different results [13]. For this reason, analysts must document their steps and reasoning if others are to understand, and ultimately trust their work. Yet, the iterative and exploratory nature of data analysis complicates documentation. Analysts struggle to track which of the many versions

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CHI 2018, April 21–26, 2018, Montreal, QC, Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5620-6/18/04 \$15.00
<https://doi.org/10.1145/3173574.3173606>

of their code produced a particular result [11, 17]. Exploration often leads to dead-ends, prompting analysts to view code as being “throw-away” and see little point in annotating it [17]. Over time analysts produce dozens of similarly named scripts, figures, and files, which can be difficult to navigate [35]. Together, these factors complicate tracking and sharing of analyses, undermining replication and review.

Computational notebooks address these problems by combining code, visualizations, and text in a single document (Figure 1). While they have ties to Knuth’s early work on literate programming [20], and have been available for decades in software such as Maple and Mathematica, the recent emergence of open-source computational notebooks has enabled rapid adoption by millions of researchers, data analysts, and journalists. Many users adopt computational notebooks with the aim to not only perform, but also document and share their analyses. Indeed, computational notebooks were designed to support construction and sharing of *computational narratives* [28]. Are they being used to share compelling narratives, or simply to explore data?

This paper presents three studies that document a tension between exploring data and explaining process and how this hinders construction and sharing of computational notebooks. We assess current use of computational notebooks through quantitative analysis of over 1 million notebooks shared online, qualitative analysis of over 200 academic computational notebooks, and interviews with 15 academic data analysts. In place of rich narratives, we find computational notebooks are often loose collections of scripts and notes, or lack explanatory text altogether. We find individual notebooks rarely tell a story by themselves but are routinely combined with other notebooks, emails, slides, and READMEs. We find many analysts see their notebooks as personal, exploratory, and messy artifacts and prefer using other media, or highly cleaned notebooks, to share results.

Making data and analyses understandable and public is crucial to advancing open science and enabling reproducibility. We believe computational notebooks are an important step towards these ends, but still in an early stage of development. Guiding their evolution requires careful examination of how they are being used, especially to better support both analysis and communication. While our results demonstrate a lack of explanation, they also highlight opportunities to foster more explanation in computational media while still encouraging the exploratory process that analysts use to think with data.

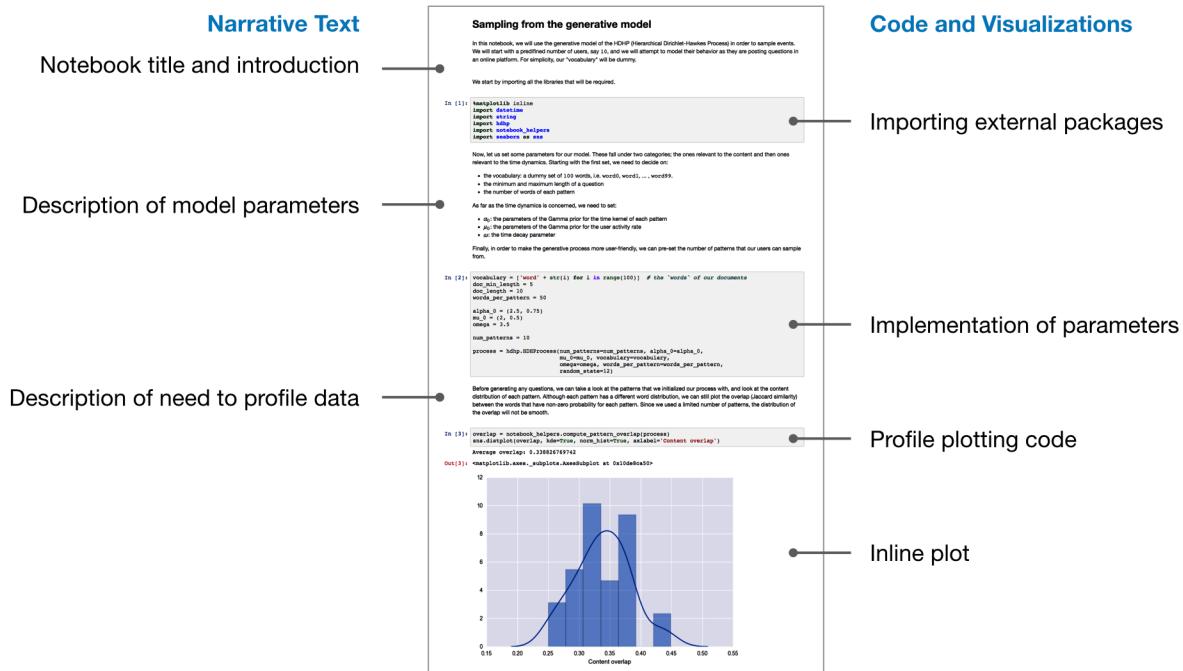


Figure 1: The first half of a computational notebook analyzed in our second study, which demonstrates a novel Python package for modeling patterns of online learning activity. The notebook combines code, visualizations, and text into a computational narrative

RELATED WORK

This research builds on three bodies of related work: understanding data analysis, the design of computational notebooks, and computational narrative.

Data Analysis

In the introduction to his seminal book *Exploratory Data Analysis* John Tukey memorably described his subject as “looking at data to see what it seems to say” [37]. The definition is vague, which may have been Tukey’s point. He stressed that exploratory analysis and plotting, preferably by hand, should almost always precede more exact statistical tests as hand-guided exploration can identify interesting trends, check assumptions, and inform selection of analytical techniques.

From large scale user data collected on the Web, to big science initiatives collecting and integrating petabytes of data, the scale of data analyzed today routinely exceeds what can be plotted by hand [22]. Still, recent accounts of data analysis echo Tukey’s description of an iterative and imprecise art. Two interview studies found that data analysis involves cycles of obtaining, cleaning, profiling, analyzing, and interpreting data [11, 17]. In their iterations, analysts try different versions of the same analysis, slowly improve analytical methods, and hit numerous “dead ends” before finding an explanation that “fits” the data.

These studies demonstrate the sensemaking [30] involved in data analysis and that the process used to collect, explore, and model data has a significant impact on the sense made. While the goal of this process may be to produce generalizable knowledge, more often than not it may simply be to

inform the analyst seeking to make a decision [19]. This can make it difficult to perform an “objective” analysis, or to precisely document or share results. Moreover, it implies that given the same dataset, two analysts may come to different, though potentially equally valid, conclusions [13].

Data analysis requires professional judgment to collect, clean, analyze, and interpret data, which has additional consequences for the ways analysts document and share their work. While observing analysts at the International Monetary Fund, Harper and Sellen found that the more judgment involved in producing a piece of information, the less suitable it was for sharing over asynchronous electronic media [12]. Analysts at the Fund routinely interpolated missing data or adjusted figures based on their knowledge of countries’ data collection practices. Without knowing how and why these adjustments were made, others could easily misinterpret the data and insights drawn from them. For these and other reasons, analysts require ways of fluidly documenting and communicating their work.

Computational Notebooks

The amount of judgment and exploration involved in data analysis necessitates clear documentation of analyses if others – or even the original analyst – are to inspect, replicate, or build on them. Leading work on reproducibility suggests that at a minimum, analysts should distribute the code used in their analyses [27]. Yet, analysts themselves may have difficulty reconstructing the exact process used to generate a result [11]. Moreover, the analysis may involve combining and reflecting on media from a variety of digital and paper resources that are not easily shared [35]. Even with all the code and resources in one place there is the

additional challenge of making them understandable. As the organizers of the Software Carpentry workshops note, “most researchers are never taught the equivalent of basic lab skills for research computing” [38]. These include placing explanatory comments at the start of every program, making code dependencies explicit, and separating raw and cleaned data. Much of this organization and annotation is a manual process learned through experience.

One way to address these challenges is to perform data analyses in computational notebooks. In the tradition of Knuth’s literate programming [20], computational notebooks enable analysts to mix code with manual annotations in a single document. While computational notebooks have been available in propriety software for decades, recent HCI research has explored mixed methods notebooks. Burrito, for example, instrumented analysts’ computers so that analytical steps were automatically recorded and could be mixed with manual annotations [11]. PRISM enabled computational biologists to mix and reflect on paper and digital media in a hybrid laboratory notebook [35].

In recent years computational notebooks have seen wide adoption thanks to the availability of open source varieties such as RNotebooks and Jupyter Notebook, which have millions of users in fields as diverse as education, finance, and the sciences [10]. This new generation of notebooks is based on cells, each of which contains rich text or code that can be executed to compute results or generate visualizations (Figure 1). These cells are linearly arranged, but can be reorganized, reshuffled, and executed in any order.

This notebook paradigm is spreading beyond data analysis to other development and visualization environments. At the time of writing, Mike Bostock was creating d3.express, an interactive notebook version of his popular D3 Javascript visualization library [2]. Distill, an online academic publisher, uses a notebook format to explain complex machine learning research [6]. Likewise, Codestrates recently demonstrated how the notebook paradigm could be used to blur the line between development and use of an application [29]. The notebook paradigm is clearly powerful and addresses key challenges of performing data analysis. But do analysts actually use them to write clear explanations of their work, or simply to support iterative analysis?

Computational Narrative

Science “begins as a story about a Possible World—a story which we invent and criticize and modify as we go along, so that it ends by being, as nearly as we can make it, a story about real life.” – Sir Peter Medawar, Induction and Intuition in Scientific Thought (1969)

One of the key features of computational notebooks is that they enable analysts to arrange code, visualizations and text in a computational narrative. While computers are good at producing and processing data, humans are much better at understanding stories. We are not experts in narrative, nor can we summarize millennia of innovation in a few para-

graphs. However, we highlight a few salient aspects of narrative as it relates to data analysis and visualization.

At its core, a narrative is a series of ordered and connected events. The Oxford English Dictionary defines narrative as “An account of a series of events, facts, etc., given in order and with the establishing of connections between them; a narration, a story, an account”. As such, a series of disjointed events is not a narrative (e.g., a twitter newsfeed), nor is a collection of related events that are not in a particular order (e.g., an affinity diagram or mood board). Narratives occur in a variety of media including audio, text, and video, each of which have their own strategies for engaging the audience and moving the story along. While some techniques, such as the “flashback” can be employed across media, others like split-screen sequences in film, are unique to particular media [30].

Since the early 2000s, there has been increasing focus on narrative and storytelling in information visualization. Gershon and Page highlighted the power of narrative to engage and convey information and suggested that information visualization employ well-established narrative techniques such as continuity editing, filling gaps, and redundancy [9]. Segel & Heer built on this foundation by developing a design space for what they called “narrative visualizations” (visualizations with a set of ordered and connected views), and identified seven distinct genres including magazine, slideshow, and comic-strip [32]. Noting the importance of the order in which data views are presented, Hullman et. al conducted multiple studies of how people sequence information visualization events, finding they tend to prefer a consistent, hierarchical structure [14, 15]. More recently, Kosara & Mackinlay highlighted the need to use different storytelling strategies in different situations with different audiences (self-running presentations, live presentations, small-group presentations) [21], and Satyanarayan & Heer demonstrated Ellipsis, a tool to support the authoring of narrative visualizations particularly for the web [31].

This prior research demonstrates the challenge of communicating exploratory data analysis, the promise of computational notebooks, and the characteristics of narrative in information visualization. However, as noted in prior research, narrative affordances and strategies differ across media and audience. It remains to be seen what types of explanation computational notebooks afford and the distinct scenarios in which analysts use them. Moreover, tools such as Ellipsis which support the construction of narratives in interactive information visualizations may not apply when crafting narrative in computational notebooks which need to not only convey insights, but also how they were generated. In the following three studies we assess the current state of narrative in computational notebooks and the challenges analysts face constructing and sharing them.

STUDY 1: COMPUTATIONAL NOTEBOOKS ON GITHUB

To examine if computational notebooks make significant use of narrative, we analyzed 1.23 million Jupyter Note-

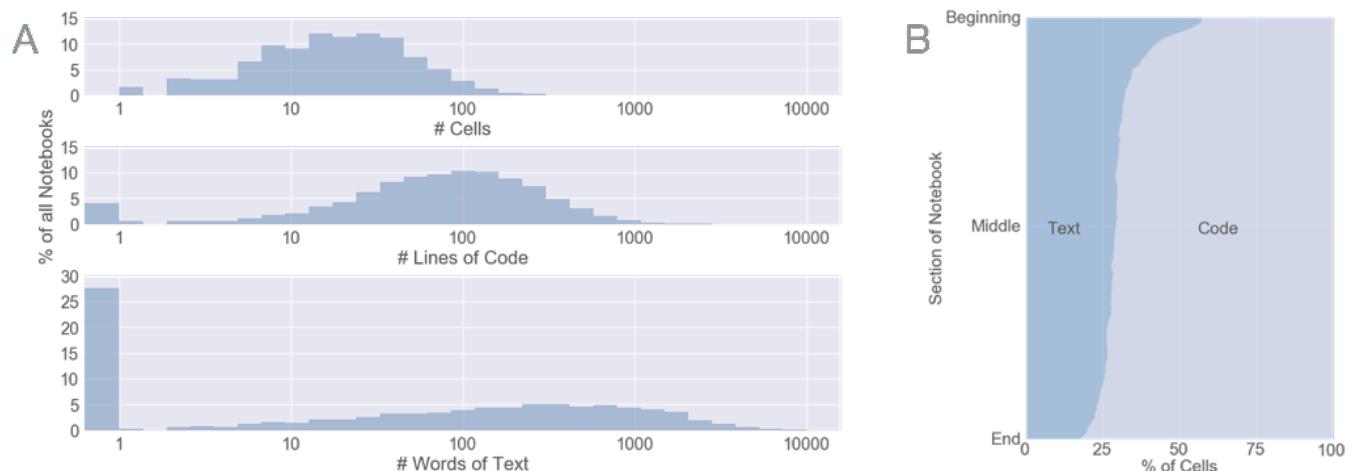


Figure 2: A) Notebook length as measured by cells, lines of code, and words of markdown. While only 2.2% of all notebooks had no code, 27.6% had no text. **B)** Content type across the average notebook. Cells at the start of the notebook were more likely to be text and cells at the end more likely to be code.

books hosted publicly on GitHub. GitHub is a popular website for hosting, managing, and collaboratively editing software source code. Jupyter Notebook is a popular open-source computational notebook that enables authors to combine code, visualizations, and text in a single document (an .ipynb file) whose underlying structure is JSON. In May 2015, GitHub began to natively render Jupyter Notebooks so that anyone viewing one on the site would see the fully rendered notebook rather than its underlying JSON [33], which has made GitHub a popular site for storing and sharing Jupyter Notebooks.

Methods

In July 2017, we searched GitHub for all publicly available Jupyter Notebooks that had not been forked (i.e., copied) from another repository (i.e., collection of code). For each notebook, we attempted to download the notebook, information about the repository that contained it, and, if present, the repository’s README. Of the 1,294,163 notebooks hosted on GitHub at the time, we were able to download notebook and repository data for 1,227,573, roughly 95% of the corpus. Unless otherwise stated, all figures in the following results are relative to this set of 1.23 million notebooks. The majority of notebooks we did not obtain data for were invalid JSON or empty files, though some data were missing because files had been moved, renamed, or deleted during our search. We computed metrics about each notebook’s content and structure to facilitate analysis.

Results

Users: There are 100,503 GitHub users who have publicly shared a notebook. This is about 0.4% of all GitHub users and 1.7% of the estimated 6 million Jupyter Notebook users [10]. The number of notebooks per user follows an exponential distribution, , with 24.5% of users hosting only one notebook on GitHub, and 27.4% hosting ten or more. The majority of notebooks (81.4%) belong to users who have hosted 10 or more.

Repositories: There are 191,402 repositories on GitHub containing at least one Jupyter Notebook. The number of notebooks per repository follows an exponential distribution, , with 39.1% of repositories having only one notebook and 14.6% of repositories having ten or more. The majority of notebooks (66.4%) belong to repositories with ten or more notebooks in them.

Language & Packages: Jupyter Notebook can execute code written in over 40 programming languages and users select a primary language when they create a notebook. Of the 85.1% of notebooks with a language specified, the vast majority were written in Python (96.3%), particularly Python 2.7 (52.5%). Notebooks written in R and Julia each accounted for about 1% of all notebooks. Of notebooks written in Python, R, or Julia, 89.1% imported external packages or modules. The most commonly imported Python packages were Numpy (67.3% of Python notebooks with imports), Matplotlib (52.1%), and Pandas (42.3%), showing a strong emphasis on data science and visualization.

Notebook Length: Jupyter Notebook cells can be any length and contain either executable code, markdown to be rendered as text, or raw content that should be rendered as is. Most cells were either markdown or code (99.8% of cells) rather than raw content. The number of cells per notebook, as well as the amount of text and the number of lines of code per notebook all followed log-normal distributions, with the exception that a significant number of notebooks (27.6%) had no text in them but consisted entirely of visualizations or code (Figure 2A). Only 2.2% of notebooks did not have any code but were entirely text. Ignoring notebooks without text, the median notebook had 218 words of text, though the longest, at 55,000 words, was longer than *The Great Gatsby*. Disregarding notebooks without code, the median notebook had 85 lines of code, though the longest had over 400,000 lines of code, more than NASA’s primary space shuttle flight software [7].

Organization: Notebooks are extremely flexible with the main organizational element being cells, which can be linearly arranged. Users can provide additional structure by deciding how to split text or code across cells, using functions, classes, and comments to structure code, and using markdown headers and links to structure text. We found that cells at the beginning of notebooks were more likely to be text, but that the majority of later cells were devoted to code. (Figure 2B). Most notebooks used headers to organize text, and comments to organize code (Table 1).

	Feature	% of all Notebooks
Text	Text	72.7
	Headers	60.2
	URLs	31.6
Code	Code	97.8
	Comments	62.1
	Functions	37.3
	Classes	12.3

Table 1: Organizational features of notebooks. Most notebooks organized text with headers, and code with comments.

Execution & Outputs: While convention is to run cells linearly from top to bottom of the notebook, cells can be executed in any order. This can be useful when checking if changes to a prior analytical step impact later computations. Jupyter Notebooks track cell execution order, so we were able to see if notebooks were run linearly or non-linearly. We found that 43.9% of notebooks with computational output had a non-linear execution order. Jupyter supports three types of output: stream (e.g., print statements), executed results (e.g., numerical results), and displayed data (e.g., rich data displays such as graphics and tables). In our corpus, 85.0% of notebooks had output in at least one cell, with 68.5% of notebooks having stream output, 58.1% having an executed result, and 45.5% having displayed data.

Description of Repositories: GitHub repositories provide a number of facilities for describing and documenting projects. These include a short description, longer README files that get rendered on the repository’s homepage, and GitHub-hosted project websites. While 58.5% of notebook repositories had a description and 73.0% had a README, only 4.5% had a GitHub-hosted project website. Analyzing the descriptions gives a sense for the topics notebooks analyze and discuss. The ten most common words in repository descriptions were *learning, project, machine, udacity, course, deep, nanodegree, neural, kaggle, and model*, showing an emphasis on machine learning and education.

Discussion

Our corpus is very diverse, including notebooks containing a single line of code and others that are full interactive textbooks. While some stood alone, others were part of a collection that formed a multi-step analysis. While some were

homework submissions, others demonstrated software packages, or documented original research. This diversity discourages generalization, but we highlight a few broad trends in the use of text and narrative.

First, most notebooks were not narratives but collections of scripts with loose notes. A quarter had no text and, even disregarding these, the median notebook had barely more text than the abstract of this paper (Figure 2A). The designers of Jupyter Notebooks intended for them to be a “Literate Computing environment, [in which] the author weaves human language with live code and the results of the code, and it is the combination of all that produces a computational narrative” [16]. It appears many analysts are not using notebooks in this way.

Second, descriptive text is not evenly distributed across notebooks (Figure 2B). Text is most likely to occur at the very beginning of the notebook, steadily less prevalent as the notebook progresses, and least likely to occur at the very end. This may reflect the use of introductory text to present the goals and organization of the notebook, but not conclusion text to reiterate goals and interpret results. Alternatively, the declining use of text as the notebook progresses may demonstrate that less explanation is needed once the analysis has been setup, or that analysts tire of annotating the notebook over time.

Third, notebooks in our corpus rarely stood alone. The vast majority were in repositories containing other notebooks, a README file, or both. A single narrative may flow across multiple notebooks, from one for data cleaning into another for profiling and modeling. Moreover README files may provide additional information about the motivations, background, and findings of the analysis.

Finally, the exploratory and iterative nature of data analysis is reflected in the fact that nearly half (43.9%) of notebooks were uploaded to GitHub with a non-linear execution order. This means that analysts went back and re-ran earlier cells, rather than just linearly writing and executing code. This figure should be considered as a lower-bound as analysts may have done a clean run of their notebook before sharing it online.

These results demonstrate that while many notebooks are used for iterative analysis, few contain lengthy explanations of their contents. Are analyses performed in notebooks being explained in other ways? Or might it be that particular uses of notebooks employ more narrative than others? We begin to address these questions by focusing on one particular community of practice: academic data analysis.

STUDY 2: NARRATIVE IN ACADEMIC NOTEBOOKS

In this second study, we focused on how notebooks documenting academic data analysis employ narrative. We selected these notebooks because, relative to other communities, the collaborative nature of academic research may favor inclusion of text to explain methods and results so others can understand and build on the work. Transparency and replicability of analytical processes is also of increasing

importance in the scientific community [26]. To give an idea of the richness of some scientific notebooks, one highlighted by the Jupyter team [16] which supplements a Nature article [5] contains over 2000 lines of code and 7000 words of text, even as the Nature article itself is half that length at 3500 words. We explore whether this example is an outlier, or if most academic notebooks employ narrative.

Methods

Sampling: We sampled academic computational notebooks by searching GitHub for repositories with both a notebook and a README linking to an academic publication. In a pilot analysis of “interesting” academic notebooks [16], we found that many notebooks were in repositories whose README had a URL pointing to a journal, conference, or pre-print publication. While many of these links lead to journal-specific websites, such as nature.com, the most common links pointed to Document Object Identifiers (DOIs) and arXiv publications (a popular pre-print service). To obtain a sample of academic computational notebooks, we searched GitHub for repositories containing Jupyter Notebooks and a README with a DOI or arXiv link. We purposefully sampled the resulting 858 repositories to get 52 from a range of disciplines, looking for keywords such as “chemistry”, “physics”, and “linguistics” in the READMEs. These 52 repositories contained 221 notebooks.

Coding: We iteratively coded all 221 notebooks to develop codes describing how academic notebooks employ text. Specifically, we coded each notebook’s genre, organization and use of text, and the organization and use of code comments (Table 2). Two researchers open coded 50 notebooks to develop initial codes and refined and reapplied these codes until they achieved greater than 60% inter-rater reliability (Cohen’s Kappa). They then divided and separately coded the remaining notebooks. They used a similar process to identify features of the repositories containing academic notebooks, coding for the contents of the repository as well as contents of the README.

Results

Repository Content and Readmes: In 43 of the 52 repositories, notebooks made up the majority of contents, averaging 81.6% of the repository’s total bytes. In the nine cases where notebook content was the minority, the majority of repository contents were program files that the notebook imported. In addition to notebooks, the majority of repositories contained source code in program files such as .py files (40). Many contained raw data (24 repositories), figures (15), manuscript files (10) and additional documentation (7). Most repository README files described what the repository’s code did (33 repositories) and the steps required to setup or install it (33). Many READMEs also described the organization of the repository’s files (24) and how to execute the code or notebooks once configured (18). Few discussed analytical reasoning (7) or results (10).

Notebooks: Half of the repositories (26) contained a single notebook, █. The two repositories with the most

notebooks (52 and 26 respectively) were largely repetitive with notebooks that tweaked one or two parameters at the top, and then ran the exact same collection of cells. To prevent these nearly identical notebooks from biasing our data, we removed them from further analysis, leaving 50 repositories with 145 notebooks. These 145 notebooks were longer than the notebooks from our GitHub corpus in Study 1 with a median length of 31 cells (compared to 18 in Study 1) 102 lines of code (85), and 329 words (218).

Organization and Use of Text: Most notebooks had an introductory text cell (55%) but almost none had a concluding text cell (3%). The vast majority used headers (86%), and slightly fewer had text aside from the headers to explain the analysis (77%). Of those notebooks with non-header text, 88% used that text to describe analytical steps, but only 34% to explain reasoning, and just 38% to discuss results.

Organization and Use of Code Comments: We found 82% of notebooks had code comments. Of these, almost all (99%) used comments to describe what the code was doing, and half (50%) used them to control the program flow by commenting out alternative code. Very few notebooks used comments to explain the analysts’ reasoning (10%) or results (4%).

	Analysis	Tutorial	Figure	All
# Notebooks	54	41	50	145
# Cells	38	25	17	31
Lines of Code	102	89	162	102
Words of Text	434	213	103	329
Headers	87%	78%	90%	86%
Text	89%	73%	66%	77%
<i>Text Intro</i>	72%	61%	30%	55%
<i>Text Steps*</i>	94%	97%	70%	88%
<i>Text Reasoning*</i>	46%	33%	15%	34%
<i>Text Results*</i>	29%	37%	48%	38%
Comments	89%	66%	88%	82%
<i>Com. Steps[†]</i>	98%	100%	95%	99%
<i>Com. Reason[†]</i>	15%	15%	2%	10%
<i>Com. Results[†]</i>	2%	7%	4%	4%
<i>Com. Flow[†]</i>	58%	37%	50%	50%

* % of notebooks with non-header text; [†] % of notebooks with code comments

Table 2: Length and content of academic computational notebook by genre. Analysis notebooks employed more text, while Figure notebooks had more code.

Notebook Genre: 54 notebooks documented a full analysis, 50 simply replicated figures, and 41 were tutorials for how to use a particular software package. The use of text varied across genre (Table 2) with full analysis notebooks more likely than figure replication notebooks to have text outside of headers, to have a textual introduction to the notebook, and to use text to describe reasoning. On the other hand, figure replication notebooks tended to use text to discuss results more than analysis notebooks. Note that due to our small sample and variance between notebooks, none of these differences were statistically significant.

Discussion

This closer examination of academic computational notebooks revealed distinct genres. Yet, even in the most verbose genre (full analysis) *analytical reasoning and results were discussed less than half the time*. While a couple notebooks contained richly detailed narratives with several thousand words of text, most were simply collections of scripts with occasional notes describing the code. Similarly, most repository README files focused on what the repository's files did and how they were organized, but did not discuss reasoning or results. This lack of explanation is not because analyses were straightforward. Even in these publicly shared notebooks, half used code comments to control program flow, demonstrating that versions of the analysis were tried, evaluated, and rejected in favor of other implementations. Notebooks were being used for iterative analyses, but not necessarily for constructing rich narratives. Indeed 90 of the 145 notebooks in our sample had less text than its repository's README.

Still, the consistent use of headers, text descriptions of steps, and README files describing repository contents demonstrates that analysts are taking time to annotate and explain their analyses. What audience do analysts consider when they annotate their notebooks? And why do they seem to devote more effort to describing steps but not higher-level motivations or reasoning? We began to address these questions in our third study.

STUDY 3: INTERVIEWS WITH DATA ANALYSTS

The second study highlighted that, when present, text in academic computational notebooks was more often used to discuss methods than reasoning or results. Seeking to better understand why these notebooks lacked rich narrative structure, we interviewed 15 academic data analysts who use computational notebooks on a regular basis.

Methods

Participants: We recruited 15 academic data analysts (4 Female, 11 Male) from eight laboratories at UC San Diego, by attending weekly lab meetings and emailing open science listservs at the university. Participants included six postdocs, five PhD students, three staff researchers, and one undergraduate student. Participants researched topics ranging from computational biology and pharmacology to astronomy and engineering science. Four laboratories had multiple people using computational notebooks as well as extensive infrastructure for running, storing, and sharing notebooks. In the other four labs, our participants were the only ones using computational notebooks. Five of our interviewees had authored at least one notebook from our Study 1 corpus, though we did not specifically recruit them for this reason. None was the author of a notebook included in our Study 2 corpus.

Procedure: We conducted twelve semi-structured interviews, three with pairs of analysts and nine with individual analysts. Each interview lasted 30–45 minutes and focused on how each analyst organized, edited, and shared computa-

tional notebooks. We grounded each interview by discussing at least one notebook the analyst had been working on recently. Sample questions included:

- Can you show us a notebook you have been working on recently?
- Can you explain the analysis in this notebook?
- What sections or cells have you spent the most time working on?
- Who else has access to this notebook? Do you plan to share it further?
- Would you need to make any changes before sharing it further?

We transcribed each interview and iteratively generated an affinity diagram to identify themes across participants.

Results

Notebook Uses: Participants used notebooks for a variety of reasons, many of which were educational. Analysts gave lectures in notebooks, assigned homework in notebooks, and used notebooks to train new lab members. While these educational uses warrant further study, we focused our interviews on the use of notebooks in research, where they were most commonly described as playgrounds for experimentation (seven participants), particularly when prototyping and debugging code. While many used notebooks to develop pipelines to automate multi-step analyses (five participants), some felt that notebooks were best for small-to-medium sized tasks and preferred language-specific development environments for larger analyses which they would run repeatedly as new data became available (two participants). Two other participants would not run analyses in the notebook but copied code into the notebook as a record of work performed elsewhere.

Analysts spoke not only of notebooks' initial use, but also their ongoing reuse. One reason for reuse was *tracking provenance*, that is, the exact steps used to generate a result. This provenance was useful for keeping track of what analyses had been tried, even if they led to dead ends, keeping older versions of figures in case an advisor decided they preferred them to the new one, and helping analysts untangle exactly how they achieved a result. While analysts can use computational notebooks to track their every step, it does not happen automatically, especially when cells are overwritten and re-run. As one participant mentioned:

I wanted Jupyter to be the tool that tracked what I did, and I'm sad that it's not - P6

A second reuse purpose our participants mentioned was *code reuse*. Individual analysts might want to reuse snippets of code from prior notebooks, or they might want to copy code for others' notebooks.

I don't necessarily want to delete that messy version of the notebook because I might not even remember if I had something in there that, like I, I might want again. – P14

A third purpose was *enabling replication*. This meant the code should be clean and annotated enough that another analyst could reasonably re-run the notebook on their machine. As one participant noted, there are numerous barriers to making notebooks both human and machine readable, and preparing notebooks for replication requires more careful construction than preparing notebooks for code reuse.

Should sharing just be, look at their code, 'Oh they did that? Maybe just that. Maybe it's too much to go all that way... it's really hard to make it runnable on somebody else's server – P13

A fourth purpose for notebooks was *presenting results*. In these notebooks analysts downplayed the role of code and added text to describe methods and results. In many cases, they even transferred outputs of the analysis to an entirely different medium (e.g., slides, word processing document) for easier review. In some cases this was for a non-technical audience, so analysts tried to draw attention away from the code and toward the conclusions by copying results to another media for sharing. In other cases, when the audience was technical and the desired feedback was technical, analysts would focus on refactoring code in the notebook so it was easy to understand and critique.

While notebooks serve these different purposes, it is difficult for them to serve more than one at a time.

It's a trade-off between having a very extensive notebook where every step is documented, or only tracking the last evolved state of whatever the question is. – P1

I know I need to make a new version of it that I think will be like, "Github ready". I want my notebook to look like the examples of notebooks that I talked about from my lab mate, but those are so clean that they don't represent my normal notebooks. They're like, presentation notebooks. Like this is perfect. This has descriptions of all the stuff I did and there's no fooling around. – P14

Sharing: Analysts shared their notebooks in ways that reflected their perspective on appropriate uses and audiences. For some analysts, notebooks were personal artifacts, best for individual use or select sharing with other technically oriented *insiders*:

A notebook is a very personal thing, so even if I would say, "Okay, here [labmate] please look into it", it wouldn't be very helpful because it's very much reflecting my style and for sure he would do slightly different types of analysis to come to the same conclusions. – P1

I think, that notebook as a medium is sort of useful to, you know, those insiders, the people that will be interested and will, you know, tweak some parameters and then possibly, you know, redo the exact same analysis just on different data. – P2

These analysts were skeptical that collaborators wanted to see their code and instead shared results in mediums such as email, word processing documents, and slides. They would often attach the full notebook if their collaborator wanted to see more details, but some felt that reviewing the notebook got in the way of interpreting higher-level findings and providing feedback.

So over time I had to realize that the collaborators... have no computer science background, nor a very strong microbiome background, so I have to report on a very high level... I try to condense what I'm finding within one sentence. So, I try to reply to mail with very few sentences. And I'm attaching the PDF [version of the notebook] should the person be interested in details, but typically no one is really looking into the methodology I'm applying, so they just trust me. – P1

I've got all this code and I've got my data but this is really not interesting and, you know, my collaborators should not really be worried about that. They should be worried about, like, what do these figures represent and whether this is something that they are expecting, or is this, is it likely there is something wrong with the way that we are processing their data. – P2

In contrast to this “notebooks for insiders” perspective, some analysts felt that notebooks were good for interacting with people who didn’t program. However, notebooks for a general audience required careful curation to make them easily interpretable. Having programming novices run the notebook could present additional challenges.

I'm trying to explain every detail unless it's like very intuitive... I wanna have chemistry people be able to read the notebook... and if there is a problem, they are going to have to look for a bioinformatician. But I just wanted them to be able to read for now. – P8

Cleaning, Layout & Annotation: Whether for personal or shared use, every analyst felt their notebooks had to be cleaned. Analysts described their notebooks as “messy”, containing “ugly code” and “dirty tricks”, and in need of “cleaning” or “polishing”. Notebooks needed cleaning because analysts were “too lazy” to add annotation, needed to be “at their best” to produce well annotated notebooks, or simply “ran out of time”.

Mine feels like a mess, mine feels like if somebody else looked at it they wouldn't have any idea what, really what order [I ran the code in], or like why I did things. – P12

Cleaning involved both organizing the notebook and adding textual annotation. Organization included adding tables of contents, sequentially numbering sections within and across notebooks, keeping scripts in individual cells under 100 lines of code, and splitting analyses that were “bulky” or “crowded” into multiple notebooks.

I like to break apart my analyses into what I consider to be notebooks that cover all the work you would do up to a stopping point where a human has to evaluate it. – P7

For me [the biggest challenge is] organization, I don't know if I should do things chronologically or if I should do things by type of data... If I run something and I run it four different times do I just make a note up here of the four parameters I used, or do I do four different cells where I ran it each time?... at some point [I end up] just getting frustrated and I'll make a new notebook. – P12

Analysts annotated their notebooks both for personal use and easing interpretation by collaborators. Personal documentation was added to prevent “getting lost” in the notebook, to remember what was done previously, and visually differentiate sections of the analysis to aid scanning.

So I try to document what I'm doing, or at least what the tasks are because it's so easy to get lost in all the different specific questions. – P1

When the notebook was to be shared, documentation focused instead on presenting the analysis at a high level, providing background information and interpreting results.

The thing that I usually end up having to put in that's tedious but it's kind of the whole point, is, you know, okay I generated these beautiful visualizations and then what are the conclusions that I drew from them, because, in our role, we're supposed to be the experts who are saying not just, “This is the visualization”, but “If you look at this visualization the conclusion that you should draw”... the interpretation, that is not something that can ever be auto generated. – P6

Social Expectations: Several analysts felt that there was not sufficient social expectation or practice to make widespread sharing and detailed annotation of notebooks feasible.

Does your PI [Primary Investigator] care about the code or not? And I think that most of the time, from my experience, it's been no. You know they just want to see the plots... That's, I think, driven by what, you know, what our old kind of standard kind of pathway was... it was a lot of just scripts that you couldn't really port and couldn't really make available. – P15

So I know with like paper lab notebooks in the wet lab you get really heavy training in like... you should write like your name, your date, kind of the hypothesis that you're doing, a little bit of the intro and then like your materials what your steps were and then some sort of conclusion like and your data. And there's a lot of that training and it's not ever, like the notebook isn't physically set up that way where it has on each page like, intro, conclusion, whatever. But you just do it that way because that's how you're trained to do it. – P12

Publication and Reproducibility: Despite this lack of training or pressure to share notebooks in some labs, many analysts expressed a desire, even an obligation to document their notebooks in such a way as they would be reproducible, that is, that they could be run by another analyst on their own computer. However our participants expressed several barriers to making their notebooks truly reproducible. One was deciding when the analysis was ready for publication. Another was receiving pushback from collaborators when preparing to publish a notebook publicly.

So to define the point in time when a publication is finalized is very complicated. Is it when you first submit? Is it when it's in review? Or when you only have to do some format editing? So there is no hard deadline unto which you have to finalize your notebook, and therefore it's very easy to not do it... it's a lot of additional work, and you also have a todo list of more pressing issues, so it's easy that this publication or making the notebook public will be missed. – P1

The couple times I've mentioned it [publishing a notebook] I've gotten people, like, they're worried that it like opens them up to more criticism than it's worth for them. – P7

Discussion

These results highlight the effort involved in organizing, annotating, and sharing computational notebooks. In particular, they highlight the tension between exploration, in which iterative experimentation tends to produce “messy” notebooks, and explanation, in which these notebooks are “cleaned” for a particular purpose (tracking provenance, code reuse, replication, presentation). In each of these roles, there is a tension between exploration and explanation. Notebooks that track provenance focus on faithfully tracking the exploratory process of data analysis. However, given the interactivity of notebooks, analysts seeking to track provenance need to be careful to not overwrite past actions. Alternatively, notebooks for presentation may obscure almost the entire exploratory process of data analysis in an effort to make it easy to review and provide feedback on the results. Notebooks to be shared publicly online had to meet an even higher standard of cleanliness that one participant noted removed all the exploratory “sandbox” material.

Finally, while notebooks enable analysts to wrap computational code and results with explanatory text, they do not necessarily prompt more frequent reflection or annotation. Social practices like presenting at lab meeting and writing papers may still be stronger triggers for these explanatory and sensemaking activities. As P12 noted:

...it's mostly lab meetings and then actually writing the paper that are the only times, or like the initial planning, that are the only times where you have to sit and be like “Why am I doing this? What am I gonna do? What am I finding? What do I think it means?”

CONCLUSION

Computational notebooks address many fundamental challenges with performing, documenting, and sharing data analyses. They support incremental and iterative analyses, enabling users to edit, arrange, and execute small blocks of code in any order. They enable explanation of thought processes by allowing analysts to intersperse code with richly formatted textual explanations. They facilitate sharing by combining code, visualizations, and text in a single document that can be posted online or emailed. Some computational notebooks are truly remarkable in the way they elegantly explain complex analyses [16].

Balancing Exploration and Explanation

Yet, the three studies in this paper demonstrate a tension between exploration and explanation that complicates construction and sharing of computational notebooks. The exploratory process of data analysis tends to produce “messy” notebooks with alternative code and duplicate cells. These notebooks need to be cleaned before they can clearly explain the analysis to a particular audience (e.g., the analyst’s future self, a technical colleague, a manager, or the public) for a particular purpose (e.g., tracking provenance, supporting code reuse, enabling replication, presenting results). Cleaning notebooks is often tedious, manual work, and it is difficult to craft notebooks that serve more than one purpose or address more than one audience at a time. Many analysts simply choose to explain and share their analyses using other, more established media, and provide a link, for the “curious”, to the notebook where they performed the analysis in the first place.

The issues of notebook “cleanliness” and intelligibility resonate with the discussion of refactoring [8, 25] and “technical debt” in software engineering. Technical debt refers to coding strategies that save time or energy in the moment but lead to extra costs later on [4]. Rather than calling for the elimination of technical debt, recent work acknowledges its inevitability and suggests better ways to manage it [1, 3]. While some lessons from this literature may apply to data analysis, there are significant differences in the process of iteratively writing scripts to analyze data and writing robust source code for enterprise applications.

Limitations

We focused on Jupyter Notebook, so our results may not reflect patterns of use typical in other varieties of computational notebook such as RNotebooks or Mathematica. However, Jupyter Notebook is currently one of the most widely used platforms and the one with the most explicit support for narrative. Our first study looked at notebooks shared publicly online, some of which may have been placed there for archival purposes, without sharing in mind. Moreover public notebooks may look different from those reserved for personal use or sharing with a small group. Lastly, our final two studies focused on academic data analysis, revealing three genres of notebooks common in that community. Future studies could broaden scope to consider other forms of notebooks in educational or enterprise contexts.

Opportunities for Design

The lack of explanation in computational notebooks may be discouraging for those who, like us, see them as a vital tool for promoting open science, reproducibility, and greater engagement with data. However, our findings also highlight opportunities for design to facilitate greater explanation in computational media without hindering exploration. One key challenge will be developing tools that augment analysts’ workflows to facilitate organization and annotation without much additional effort, either by piggybacking on existing user behavior or by providing an immediate benefit to users’ more active annotation or organization activities.

Leverage structure. One approach could leverage explicit annotation to aid navigation, debugging, or checking the status of variables or processes. For example, one interviewee began annotating his notebooks more after he installed an extension that would float a table of contents over his notebook based on its markdown headers, easing navigation. Such structure could also be used to automatically generate documentation or comments [23, 34].

Encourage best practices. Notebooks could make it easier to follow computational best practices, such as those taught in Data Carpentry workshops, or automatically flag “code smells” [8, 36]. For example, a notebook “clean-up” tool could encourage moving imports to the start of the notebook and rewriting reusable code as functions, which would improve maintainability and legibility in the long run.

Support non-linear narrative. Notebooks’ linear structure is elegant, but may not fully support the analytical process. Automated version control or more sophisticated “layers” could simplify or enrich notebooks’ narrative structure.

Social interventions. More journals could incentivize publishing well-annotated analysis code alongside results and individual labs could develop new methods and norms of sharing and reviewing preliminary results. As it is, one interviewee noted that many of his labmates seemed “put-off” by seeing presentations from a notebook, possibly feeling that the presenter had not taken any time to prepare.

In the end, it will take a combination of technical and social factors to encourage clearer explanation of data analyses and more structured ways of supporting explorations in computational media. Computational notebooks are a young media compared with paper notebooks, whiteboards, and journal articles, or even other programmable media such as apps or web pages. It will require time, experimentation, and inventiveness to develop tools and practices surrounding computational notebooks that facilitate greater explanation and sharing of the iterative, imprecise, and messy art that is data analysis.

ACKNOWLEDGEMENTS

We thank Regina Cheng and Nathan Hassanzadeh for their help collecting and analyzing data for Studies 2 and 3. This research was funded by NSF grants #1319829 and #1735234 as well as NLM grant #T15LM011271.

REFERENCES

1. Eric Allman. 2012. Managing Technical Debt. *Communications of the ACM*, 55, 5 (May 2012), 50-55. DOI: <https://doi.org/10.1145/2160718.2160733>
2. Mike Bostock. 2017. A Better Way to Code. (Apr. 2017). Retrieved September 11, 2017 from <https://medium.com/@mbostock/a-better-way-to-code-2b1d2876a3a0>
3. Nanette Brown, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, Nico Zazworka, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord. 2010. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of Software Engineering Research (FoSER '10)*. ACM Press, New York, NY, 47-52. DOI: <https://doi.org/10.1145/1882362.1882373>
4. Ward Cunningham. 1992. The WyCash portfolio management system. *Addendum to the proceedings on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '92)*. ACM Press, New York, NY, 29-30. DOI: <https://doi.org/10.1145/157709.157715>
5. Tao Ding and Patrick Schloss. 2014. Dynamics and associations of microbial community types across the human body. *Nature*, 509, 7500 (Apr. 2014), 357-360. DOI: <https://doi.org/10.1038/nature13178>
6. Distil. 2017. Retrieved September 11, 2017 from <https://distill.pub/>
7. Brian Dunbar. 2010. NASA - Shuttle Computers Navigate Record of Reliability. (June 2010). Retrieved September 15, 2017 from https://www.nasa.gov/mission_pages/shuttle/flyout/flyfeature_shuttlecomputers.html
8. Martin Fowler. 1999. Refactoring: improving the design of existing code. Addison-Wesley Professional, Reading, MA.
9. Nahum Gershon and Ward Page. 2001. What storytelling can do for information visualization. *Commun. ACM* 44, 8 (Aug. 2001), 31-37. DOI: <https://doi.org/10.1145/381641.381653>
10. Brian Granger, Chris Colbert, and Ian Rose. 2017. JupyterLab: The next generation jupyter frontend. *JupyterCon 2017*.
11. Philip Guo and Margo Seltzer. 2012. Burrito: Wrapping your lab notebook in computational infrastructure. *USENIX Workshop on the Theory and Practice of Provenance (TaPP '12)*.
12. Richard Harper and Abigail Sellen. 1995. Collaborative tools and the practicalities of professional work at the international monetary fund. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '95)*. ACM Press, New York, NY, 122-129. DOI: <https://doi.org/10.1145/223904.223920>
13. Thomas Herndon, Michael Ash, and Robert Pollin. 2014. Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Cambridge journal of economics*, 38, 2 (Dec 2013), 257-279. DOI: <https://doi.org/10.1093/cje/bet075>
14. Jessica Hullman and Nicholas Diakopoulos. 2011. Visualization rhetoric: Framing effects in narrative visualization. *IEEE transactions on visualization and computer graphics*, 17, 12 (Dec. 2011), 2231-2240. DOI: <https://doi.org/10.1109/tvcg.2011.255>
15. Jessica Hullman, Steven Drucker, Nathalie Riche, Bongshin Lee, Daniel Fisher, and Eytan Adar. 2013. A deeper understanding of sequence in narrative visualization. *IEEE Transactions on visualization and computer graphics*, 19, 12 (Dec. 2013), 2406-2415. DOI: <https://doi.org/10.1109/tvcg.2013.119>
16. Jupyter. A gallery of interesting Jupyter Notebooks. Retrieved September 11, 2017 from <https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks>
17. Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2012. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18, 12 (Dec. 2012), 2917-2926. DOI: <https://doi.org/10.1109/tvcg.2012.219>
18. Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: supporting exploratory programming by data scientists. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM Press, New York, NY, 1265-1276. DOI: <https://doi.org/10.1145/3025453.3025626>
19. Alison Kidd. 1994. The marks are on the knowledge worker. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '94)*. ACM Press, New York, NY, 186-191. DOI: <https://doi.org/10.1145/191666.191740>
20. Donald Knuth. 1984. Literate programming. *The Computer Journal*, 27, 2 (Feb. 1984), 97-111. DOI: <https://doi.org/10.1093/comjnl/27.2.97>
21. Robert Kosara and Jock Mackinlay. 2013. Storytelling: The next step for visualization. *Computer*, 46, 5 (May 2013), 44-50. DOI: <https://doi.org/10.1109/mc.2013.36>
22. Kateryna Kuksenok, Cecilia Aragon, James Fogarty, Charlotte P. Lee, and Gina Neff. 2017. Deliberate Individual Change Framework for Understanding Programming Practices in four Oceanography Groups. *Journal of Computer Supported Cooperative Work*, 26, 4-6 (May 2017), 1-29. DOI: <https://doi.org/10.1007/s10606-017-9285-x>

23. Paul McBurney and Collin McMillan. 2014. Automatic documentation generation via source code summarization of method context. In *Proceedings of the 22nd International Conference on Program Comprehension (ICPC '14)*. ACM Press, New York, NY, 279-290. DOI: <https://doi.org/10.1145/2597008.2597149>
24. Peter Medawar. 2008. Induction and Intuition in Scientific Thought. Routledge.
25. Emerson Murphy-Hill, Chris Parnin, and Andrew Black. 2012. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, 38, 1 (Jan. 2012), pp.5-18. DOI: <https://doi.org/10.1109/tse.2011.41>
26. Nature. 2017. Announcement: Transparency Upgrade for Nature Journals. *Nature*, 534, 7645 (Mar 2017), 288. DOI: <https://doi.org/10.1038/543288b>
27. Roger Peng. 2011. Reproducible research in computational science. *Science*, 334, 6060 (Dec. 2011), 1226-1227. DOI: <https://doi.org/10.1126/science.1213847>
28. Fernando Perez and Brian Granger. 2015. Project Jupyter: Computational Narratives as the Engine of Collaborative Data Science. Retrieved September 11, 207 from <http://blog.jupyter.org/2015/07/07/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science/>
29. Roman Radle. 2017. Codestrates: Literate Computing with Webstrates. In *Proceedings of the ACM on User Interfaces and Systems Technology (UIST '17)*. ACM Press, New York, NY.
30. Dan Russell, Mark Stefik, Peter Pirolli, and Stuart Card. 1993. The cost structure of sensemaking. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '93)*. ACM Press, New York, NY, 269-276. DOI: <https://doi.org/10.1145/169059.169209>
31. Arvind Satyanarayan and Jeffrey Heer. 2014. Authoring narrative visualizations with ellipsis. *Computer Graphics Forum*, 33, 3 (Jun. 2014), 361-370. DOI: <https://doi.org/10.1111/cgf.12392>
32. Edward Segel and Jeffrey Heer. 2010. Narrative visualization: Telling stories with data. *IEEE transactions on visualization and computer graphics*, 16, 6 (Nov. 2010), 1139-1148. DOI: <https://doi.org/10.1109/tvcg.2010.179>
33. Slava Shirokov. 2015. GitHub + Jupyter Notebooks = <3. (May 2015). Retrieved September 15, 2017 from <https://github.com/blog/1995-github-jupyter-notebooks-3>
34. Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, K. Vijay-Shanker. 2010. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering (ASE '10)*. ACM Press, New York, NY, 43-52. DOI: <https://doi.org/10.1145/1858996.1859006>
35. Aurélien Tabard, Wendy Mackay, and Evelyn Eastmond. 2008. November. From individual to collaborative: the evolution of prism, a hybrid laboratory notebook. In *Proceedings of the ACM conference on Computer Supported Cooperative Work (CSCW '08)*. ACM Press, New York, NY, 569-578. DOI: <https://doi.org/10.1145/1460563.1460653>
36. Michele Tufano, Fabio Palomba, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Andrea De Lucia, Denys Poshyvanyk. 2015, May. When and why your code starts to smell bad. In *Proceedings of the 37th International Conference on Software Engineering*. IEEE Press, 403-414. DOI: <https://doi.org/10.1109/icse.2015.59>
37. John Tukey. 1977. Exploratory data analysis. Pearson.
38. Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy Teal. 2017. Good enough practices in scientific computing. *PLoS computational biology*, 13, 6 (Jun, 2017). DOI: <https://doi.org/10.1371/journal.pcbi.1005510>