Visualization Notebooks are Paper Tools

Jasmine Tan Otto

ABSTRACT

This paper argues for computational notebooks as examples of paper tools. Through generative theories of interaction, I read notebooks as sites of creative data-driven visualization practices, and as sites where visualization literacy can be taught. Notebooks are both generative pipelines and sites of collaboration with data. Notebooks enable reflective practitioners to discover connections between multiple levels of abstraction through self-directed visualization processes. The sociotechnical characteristics of visualization notebooks enable them to function as paper tools, supporting rapid iteration on visualization design tasks by both experts and trainees. Ultimately, specific paper tools enable us to define criteria for visualization literacy.

1 INTRODUCTION

This paper identifies visualization literacy as a desiderata of *programming tools* which are used to create visualization systems. To make this argument, I will recruit 1) theories of technical literacy, focusing on *paper tools* and *instrumental interaction*, which will be reflected in successful visualization designs; and 2) existing programming environments, especially *computational notebooks*, which serve expert and trainee visualization developers' *reflective design* needs. Ultimately, this paper draws novel connections between prior theories of computer-supported collaboration and interaction design, through concrete examples of end-user programming in a visualization research context.

I will argue that computational notebooks *are paper tools* (Figure 1) per Klein's original definition [7]. This argument will be broken into three parts, as follows:

(1) Computational notebooks are *a medium of communication* that supports rapid prototyping and allows practitioners to readily share tentative visualization results, supporting deeper conversations (Sec 3.1).
(2) Computational notebooks evolved from the *genre of tools* used as programming environments in response to the needs of visualization practitioners, including literate visualization (Sec 3.2).
(3) Computational notebooks are named for **lab notebooks**. They are artifacts which document prior explorations, and for the author, serve a reflective purpose. Notebooks are both tool and mediator, as they can be shared with collaborators and students to share complex insights (Sec 3.3).

As to the question of how visualization literacy may be evaluated *as an outcome* of paper tool use, I direct interested readers to the **generative theories of interactivity** presented in an interaction design context by Beaudouin-Lafon et al. [1], which I discuss in Section 2.2.

2 BACKGROUND

This section introduces paper tools as a key concept underlying visualization literacy. I begin by describing research processes of the greatest generality, through considerations of interactive visualization systems, and finally to specific desiderata in various existing programming tools used to develop visualization systems.

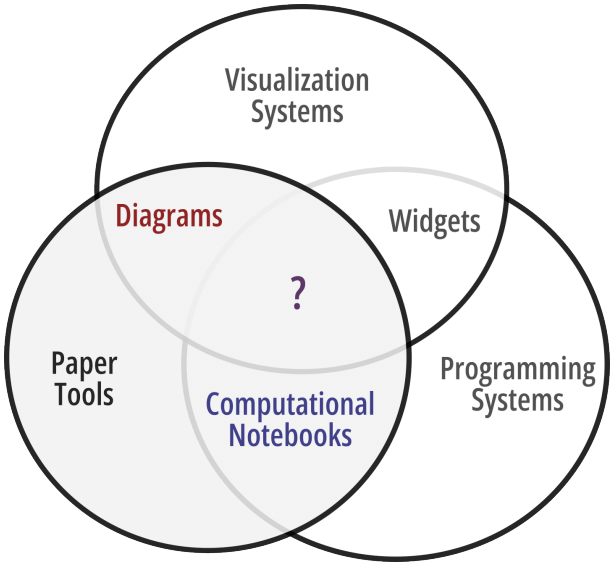

Figure 1: Genres of artifact which participate in reflective sense-making processes. By distinguishing *paper tools* from other forms of visualization and/or programming systems, I intend to orient conversations about visualization literacy towards *what falls in the center*.

2.1 Paper tools

Rapid prototyping is a strategy for *critical reflection* [13], a kind of design process that we see in the world of research (and especially scientific research) quite often. Our discussion therefore begins with *paper tools* [7], pre-existing systems of notation which participate in complex arrangements of scociotechnical knowledge, which will ground our proposed definition of visualization literacy in prior theory from science and technology studies.

Paper tools are systems of notation used by scientists to do their work and to teach their students. These notations are specialized sign systems with non-arbitrary relationships to their referents, which serve essential purposes such as recording the state of an object-of-study before and after an experimental intervention. Klein's central example is the Berzelian notation for chemical reactions developed in the 1830s, which reflects the then-recent discovery that reaction masses are preserved (in terms of the atomic elements), but may be transformed between different compounds in different proportions. This recently-invented notation (relative to other languages) is so ubiquitous now that the formula H^2O (read as 'two hydrogens and one oxygen') may be used synonymously with 'water'.

Paper tools are key to technical literacy, as evinced by subsequent articulations of drawing and writing as 'productive epistemic

practices' [4]. Like Klein, Hoffman and Wittman respond to Latour's observations of laboratory life when they argue that *signs may be manipulated like phenomena*, and therefore sign systems with this property merit study as both systems of communication and *simultaneously as* scientific instruments.

2.2 Generative theories of interaction

Hinrichs et al. defend the visualization system as a 'sandcastle' [3], i.e. not as a robust technical system upon which other software may be implemented, but rather as an instrumental tool supporting novel forms of interaction with unusual and one-off datasets. It is in this spirit that I introduce two sets of lenses on interactivity from Beaudouin-Lafon et al., who call these each 'generative theories'.

First, studying an interactive artifact through the lens of *instrumental interaction* requires reifying **abstract operations** on conceptual objects to a set of affordances provided by the artifact. This generative theory involves a set of questions which may tease out the relationship between a given artifact and its object. For example, one may ask which operations are or should be reified by a given visualization artifact. The majority of these questions apply readily to paper tools, which are interfaces comprised of signs that permit interaction with phenomena. In the domain of games studies, *operational logics* ask the same question in reverse [10]: given a certain interaction with the controls of a game, what is its action upon the game-world?

Second, studying an interactive artifact through the lens of the *community and common object* requires the identification of stakeholders and their needs, which often differ within a community. This poses an essential difficulty in defining *visualization literacy*: is literacy the ability to validate an argument, or to formulate a new one? How might we build tools to support visualization fluency, which Hinrichs et al. set out as the goal of visualization and visualization-mediated collaboration?

2.3 Visualization notebooks

I have now articulated some desiderata in visualization artifacts, and in practices of data visualization. Finally, I will establish the direct influence that programming environments have upon visualization practice, such that the *computational notebook* is a working site of visualization literacy. Reflective design asks us to adopt this frame of *critical technical practice* as a method of knowing which technical alternatives embody certain unconscious values; in this case, how notebooks embody visualization literacy.

Computational notebooks are a family of programming systems descending from the values of a) literate programming and b) literate visualization, intended to support collaborative work and decision-making per Kosara's account [8]. The description that Sedlmair et al. give of data-driven design studies [12] is a good description of how notebooks are used by visualization practitioners to rapidly iterate on small chunks of code performing intermediate stages of a visualization transform upon large or complex datasets. For our purpose, notebooks include any code editing environment which a) maintains program state across multiple chunks of live code, i.e. incorporates a read-evaluation-print loop (REPL), and b) handles intermediate outputs with graphical and/or interactive representations, such as inline plots and object inspectors.

A notebook is a dynamic visualization in the hands of the author - and through bespoke interface elements, potentially any user, - which documents a certain object-of-study. These objects may range from the climate evolution of a certain region of coast, to the interface of a certain hosted language model, or anything in between. Following Bruggerman et al., these presumed 'objects' are in fact systems with different parts that may be juxtaposed in a 'fold'. Moreover, the design study 'core' stages of *discovery, design, implementation, and deployment* are a period where the visualization practitioner chooses between many possible 'transitions' between multiple representations of the same dataset. These folds comprise the basic unit of analysis, e.g. when a notebook is used to compare multiple years of sea temperature change in different regions over a given period of time. The notebook is, poetically speaking, a viewport onto an object-in-motion.

3 LITERACY THROUGH VISUALIZATION

This section connects visualization systems, to be understood as paper tools, with a definition of visualization literacy rooted in the critical technical practices of domain experts.

3.1 Rapid prototyping with paper tools

As mediators, visualization notebooks allow novices to learn from code snippets by manipulating them in a live environment with real data. This approach to programming literacy merely requires buy-in from teachers who themselves employ programming practices in their research, rather than active intervention with students from outside of the domain.

Consider an analogy between generative pipelines in visualization authoring, and modular synthesis in music creation. It's not just the connections between the modules, and it's not just designing new modules, but familiarity with the interfaces of existing modules that enables literacy and fluent expression. And it is not enough for all of the modules to be bar charts and scatter plots; what of the ternary plots used in geology for mineral compositions [16], or the multi-scale gene-gene expression matrices used in molecular biology [15]?

3.2 Visualization literacy through notebooks

This section briefly surveys coding practices in data science, so far as they are relevant to reflective design processes involving the use of computational notebooks as paper tools. For a high-level quantitative analysis of how scientists use computational notebooks to perform exploratory data analysis and communicate narratives to stakeholders, consult Rule et al. [11]. This section establishes critical distinctions between computational notebooks and other programming environments which lack their paper tool-like characteristics. At a technical level, in software engineering terms, these features include code encapsulation and reactive update patterns. By studying these technical features of the notebook environment that are designed to aid developers, I propose to gain insight into how visualization notebooks are able to succeed as paper tools, rather than degrading into buggy code.

3.2.1 Code encapsulation. Computational notebooks are a document format for scripting languages. Code is written in *cells* which encapsulate intermediate results, like an Excel spreadsheet in one

spatial dimension. Thanks to encapsulation, the state of a spreadsheet is summed up by what the programmer sees in the cells of a given sheet (more or less). In our anecdotal experience, notebooks with smaller amounts of *hidden state* are likewise more suitable for sharing complex procedural knowledge and *enabling replication* by other data scientists (per Rule et al.).

By reading Bruggerman et al. from a software engineer's perspective, I observe that common debugging techniques (like breakpoints and print-line state dumps) are also **folds**, because they are used to reconstruct hidden processes in order to diagnose unforeseen faults. In visualization notebooks, where graphics are commonly produced as the inline outputs of code cells, folds made in the course of debugging may now take on familiar shapes like Cartesian plots, planar graphs, and other suitable visual representations of somewhat fallible data products.

Visualizations are particularly easy to produce from *map-filter-reduce pipelines*, which are commonly used to pre-process data by modifying the schema of each row (maps), which rows are included (filters), or by summarizing the data (reductions). These three higher-order functions are the major building blocks of *functional programs*, which contain as little 'mutable state' as possible in order to simplify debugging.

3.2.2 Reactive update patterns. Armed with encapsulation and functional programming, I now describe individual visualization notebooks in terms of the *data flow* between cells. A data flow proceeds from the data source through some pre-processing steps to an output cell, typically a visualization of some sort. Interactive features complicate this situation; suppose a parameter controls one of the visualization's spatial, color, or temporal scales. This input would be collected from the user by another widget in an *upstream cell*.

In reactive notebooks, cells detect their own upstream cells and resolve the dependency automatically: if that upstream cell changes and produces a different value, then all downstream cells re-evaluate themselves to reflect that change. In a visualization context, this allows linked brushing operations [9] to be implemented naturally, as if the original Snap prototype were embedded into the programming environment. Suppose the upstream cell represents a range in one axis, and the downstream cell plots all data points falling within that range; then these coordinated widgets implement a pan-zoom operation.

When downstream components update 'for free', then strange visualization prototypes ('how dense can this plot type be before it turns unreadable?') are faster to build, and more design iterations can be performed in the same study period. Suppose the visualization practitioner then adds some pointer event handling; now any visualization cell becomes an input widget for a user-generated selection on that data.

Direct manipulation [6], it turns out, is a desirable feature in most paper tools per the lens of instrumental interaction. Unlike linked brushing, it requires breaking data-flow, because unless the user-generated selection is re-incorporated into the dataset *without* modifying it 'upstream', the visualization cell will be reloaded - destroying the interactive selection. While reactivity does a lot of implementation work for free, it is not trivial to work with.

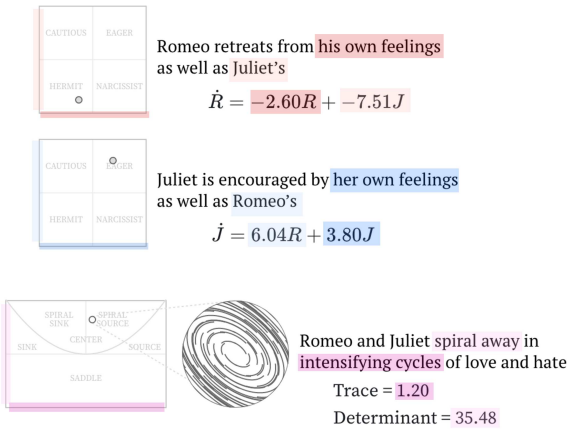

Figure 2: Two parameter input cells in *Tales from the Romeo and Juliet Phase Space* (in red and blue), and a downstream analysis cell (in purple). The linear dynamical system corresponds with a point in the trace-determinant plane, which Tucker plots alongside a trace of the system.

3.3 An interactive essay case study

I present a case study of a visualization notebook, which I identify as a paper tool. Toph Tucker's *Tales from the Romeo & Juliet Phase Space*[1] is an interactive essay written and distributed as a reactive visualization notebook. It describes a linear dynamical systems model of 'the pursuer and the pursued' as a phenomenon in stories about romance. After presenting a simulation of this model as an interactive toy for the reader to manipulate, with bespoke stability analysis tools provided, Tucker finally recruits this model to serve in a process of personal reflection via conversational prose.

Tales' content selection algorithm is based on the analysis of two-variable linear differential equations. John Gottman, who applied dynamical systems to clinical couples counselling [2], traces the metaphor to von Bertalanaffy's book on systems theory. The state of the Romeo-Juliet dyad is characterized as a pair of scalar values, which encode both affect (positive or negative) and intensity. The response of both Romeo and Juliet to their respective feelings is linear. Their romance is characterized in four scalar values, and has a closed-form solution. As Tucker puts it, "nobody can come between them — and nothing is anyone's fault but theirs."

Figure 2 depicts the parameter selection interface in *Tales*, which Tucker has designed as a literate visualization which explains the purpose of each parameter. In the figure, Romeo's approach to the relationship is highlighted in a different color from Juliet's approach, each of which the user determines by clicking in the box divided into four parts. The third color highlights the inevitable and inescapable outcome of the *linear dynamical system*, which Tucker explains is determined via stability analysis.

Figure 3 depicts the narrative generation interface in *Tales*. Tucker prompts the user to 'drop Romeo and Juliet into their story' by clicking any starting point (the initial condition), an interaction

[1] https://observablehq.com/@tophtucker/tales-from-the-romeo-and-juliet-phase-space

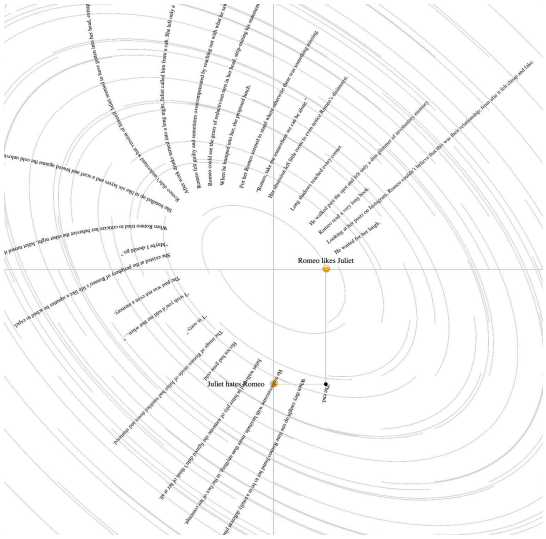

Figure 3: One possible traversal of phase space by Romeo (x) and Juliet (y). Short passages of text are randomly selected at certain intervals along their trajectory. Tucker renders the generated prose both inline with the essay, and dynamically overlaid onto this plot.

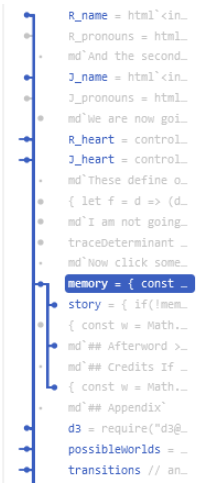

Figure 4: ObservableHQ's minimap feature, displaying an egocentric view of dataflow between cells. One cell labelled 'memory' is highlighted, whose visual output is pictured in Figure 3. The upstream dependencies of this cell are shown by the leftmost of two vertical lines, including input parameters (`R_heart` etc.) and datasets (`possibleWorlds` and `transitions`). Cells dependent on this cell's data output are shown by the rightmost vertical line: these are a) a text transcript of the trace (`story`), b) the afterword of the essay, which references the current state of the story, and c) a view of the same trace as a time-series in two variables.

which transforms an ordinary phase plane plot into a paper tool for generating romance stories. Each time the user makes this choice, a new story is generated from that new starting point. But *Tales* also encourages the user to change the four response coefficients which determine the system.

Once modified, the new coefficients propagate through the rest of *Tales*, rewriting the phase plane, which can send whatever story is currently being simulated off in an unexpected direction. The prose output of *Tales* describes the moods of Romeo and Juliet as they spin into, around, or away from the (current) phase space attractor, according to mathematical inevitability. Tucker describes their generative prose as a pastiche of various poets and playwrights, produced under a set of constraints with substantial philosophical implications: "writing characters with no identity except their conditions and how they react to them tests the principle that everyone else is just *you*, at a different time."

At first glance, visualization notebooks may appear to offer two affordances: writing code, and reading its output. We can now observe that in practice, the act of architecting a data flow allows the fluent visualization practitioner to bring in a myriad of assumptions, claims, and artifact types. *Tales* is a visualization artifact which evinces the back-and-forth dynamic between structured input and structured output. It is constructed in a way that suggests a playful and reflective process of design, and yet the final artifact describes a process of understanding a very simple and limited model.

4 PROVOCATION

Visualization literacy, understood as fluent expression in a language of analysis, requires sophisticated strategies of iterative design and mid-process reflection. Therefore, it is critical that the visualization community should consider the needs of educators who in practice use visualization strategies to communicate non-visualization research, and therefore teach visualization literacy.

I invite the reader to consider what kinds of paper tools they use in their own work. Although teaching methods involving programming systems are distinguished from similar methods, otherwise mediated - for example, a mathematics teacher talking through each step of a complex proof with sketched formulae on a blackboard - the *act of seeing a tool being used* is vital and common to both.

Computational notebooks are a specialized kind of programming system which facilitates rapid in-situ prototyping, acting much like spreadsheets with access to modern visualization libraries. Their distinct features may be understood as pointing towards the future of end-user programming in highly collaborative and creative forms of work. Next-generation programming systems related to notebooks include the interactive database-linked whiteboards of Ink and Switch [5], and Dynamicland programs which use physical objects as hot-swappable input sources [14].

Just as paper tools are both sign and phenomenon, visualization artifacts are both mediator and tool, and this dual nature is essential to their function. In this paper, I have argued that visualization notebooks are uniquely suitable for both researchers and educators to transmit visualization literacy; just as paper tools have served for centuries to externalise and transmit powerful reading strategies and information literacy.

ACKNOWLEDGEMENTS

Thanks to Toph Tucker for creating *Tales...*; to Isaac Karth and Adam Smith for their input on applications of this case study to narrative design and functional-reactive programming; and to Pablo Donato for sharing the generative theories of interaction work.

REFERENCES

[1] BEAUDOUIN-LAFON, M., BØDKER, S., AND MACKAY, W. E. Generative Theories of Interaction. *ACM Transactions on Computer-Human Interaction 28*, 6 (Dec. 2021), 1–54.

[2] GOTTMAN, J., SWANSON, C., AND MURRAY, J. The mathematics of marital conflict: Dynamic mathematical nonlinear modeling of newlywed marital interaction. *Journal of Family Psychology 13*, 1 (1999), 3–19. Place: US Publisher: American Psychological Association.

[3] HINRICHS, U., FORLINI, S., AND MOYNIHAN, B. In defense of sandcastles: Research thinking through visualization in digital humanities. *Digital Scholarship in the Humanities 34*, Supplement_1 (Dec. 2019), i80–i99.

[4] HOFFMANN, C., AND WITTMANN, B. Introduction: Knowledge in the Making: Drawing and Writing as Research Techniques. *Science in Context 26*, 2 (June 2013), 203–213.

[5] HOROWITZ, J., KALISKI, S., AND LINDENBAUM, J. Inkbase: Programmable Ink. In *Seventh Workshop on Live Programming (LIVE 2021)* (Chicago, Nov. 2022).

[6] HUTCHINS, E. L., HOLLAN, J. D., AND NORMAN, D. A. Direct Manipulation Interfaces. *Human–Computer Interaction 1*, 4 (Dec. 1985), 311–338.

[7] KLEIN, U. Paper tools in experimental cultures. *Studies in History and Philosophy of Science Part A 32*, 2 (June 2001), 265–302.

[8] KOSARA, R. Notebooks for Data Analysis and Visualization: Moving Beyond the Data. *IEEE Computer Graphics and Applications 43*, 1 (Jan. 2023), 91–96.

[9] NORTH, C., AND SHNEIDERMAN, B. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proceedings of the working conference on Advanced visual interfaces* (New York, NY, USA, May 2000), AVI '00, Association for Computing Machinery, pp. 128–135.

[10] OSBORN, J. C., WARDRIP-FRUIN, N., AND MATEAS, M. Refining operational logics. In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (Hyannis Massachusetts, Aug. 2017), ACM, pp. 1–10.

[11] RULE, A., TABARD, A., AND HOLLAN, J. D. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, Apr. 2018), CHI '18, Association for Computing Machinery, pp. 1–12.

[12] SEDLMAIR, M., MEYER, M., AND MUNZNER, T. Design Study Methodology: Reflections from the Trenches and the Stacks. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (Dec. 2012), 2431–2440.

[13] SENGERS, P., BOEHNER, K., DAVID, S., AND KAYE, J. J. Reflective design. In *Proceedings of the 4th decennial conference on Critical computing: between sense and sensibility* (Aarhus Denmark, Aug. 2005), ACM, pp. 49–58.

[14] VICTOR, B., IANNINI, L., AND DOUGLAS, S. Communal Computing for 21st-Century Science, Apr. 2023.

[15] WU, Y., ZHU, X., CHEN, J., AND ZHANG, X. EINVis: A Visualization Tool for Analyzing and Exploring Genetic Interactions in Large-Scale Association Studies. *Genetic Epidemiology 37*, 7 (2013), 675–685. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/gepi.21754.

[16] YE, C., HERMANN, L., YILDIRIM, N., BHAT, S., MORITZ, D., AND DAVIDOFF, S. PIXLISE-C: Exploring The Data Analysis Needs of NASA Scientists for Mineral Identification, Mar. 2021. arXiv:2103.16060 [physics].