Streamlining Visualization Authoring in D3 Through User-Driven Templates

Hannah Bako* University of Maryland
Alisha Varma† University of Maryland
Anuoluwapo Faboro‡ University of Maryland
Mahreen Haider§ University of Maryland
Favour Nerrise¶ University of Maryland
Bissaka Kenah‖ University of Maryland
Leilani Battle** University of Washington

ABSTRACT

D3 is arguably the most popular tool for implementing web-based visualizations. Yet D3 has a steep learning curve that may hinder its adoption and continued use. To simplify the process of programming D3 visualizations, we must first understand the space of implementation practices that D3 users engage in. We present a qualitative analysis of 2500 D3 visualizations and their corresponding implementations. We find that 5 visualization types (Bar Charts, Geomaps, Line Charts, Scatterplots, and Force Directed Graphs) account for 80% of D3 visualizations found in our corpus. While implementation styles vary slightly across designs, the underlying code structure for all visualization types remains the same; presenting an opportunity for code reuse. Using our corpus of D3 examples, we synthesize reusable code templates for eight popular D3 visualization types and share them in our open source repository. Based on our results, we discuss design considerations for leveraging users' implementation patterns to reduce visualization design effort through design templates and auto-generated code recommendations.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools—Visualization toolkits;

1 INTRODUCTION

Visualization languages are a popular tool for creating interactive visualizations [23], where D3 [5] is one of the most well-known and expressive libraries for creating web based visualizations [1, 10, 11, 13]. However, D3 is notoriously complex, often requiring extensive programming experience to use [23, 25]. Though simpler languages (e.g., Vega-Lite [26]) and direct manipulation tools (e.g., Data Illustrator [20] and Lyra [7, 33]) are available, they do not solve the user's problem if their interests are specific to D3. For example, the user may be required to use D3 at work, or they might be interested in adapting a specific D3 example found online that is not supported by less expressive tools. These users can't pivot to simpler tools or languages, and still need easier ways to program D3 visualizations.

Easing the D3 implementation process requires that we first gain an understanding of *what visual and interactive design patterns people frequently use when programming D3 visualizations*. Such information can explain users' goals when creating a visualization and what they might struggle with in the process. A plethora of D3 examples are available on repositories such as Bl.ocks.org [3] and GitHub [24] and services like Observable [4], which could enrich our understanding of users' implementation processes.

*e-mail: hbako@cs.umd.edu
†e-mail:alishav@terpmail.umd.edu
‡e-mail:afaboro@terpmail.umd.edu
§email:mhaider1@terpmail.umd.edu
¶email:fnerrise@terpmail.umd.edu
‖email:bkenah@terpmail.umd.edu
**e-mail:leibatt@cs.washington.edu

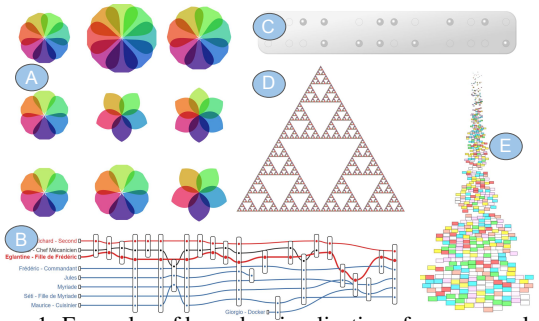


Figure 1: Examples of bespoke visualizations from our analysis. (A) renders the number of IMDB votes and corresponding ratings of movies in a movies dataset. (B) is a narrative chart of scenes from Star Wars: Episode IV. (C) visualizes a braille clock, (D) is a D3 rendering of Sierpinski Charlet, and (E) is a rendering of bounding box collisions using D3's force simulation.

In this paper, we present an analysis of D3 code repositories and example galleries, highlighting common patterns to programming D3 visualizations. We mined and qualitatively analyzed 2500 examples from GitHub, Bl.ocks.org, and Observable. To identify common strategies employed by D3 users, we analyze prevalent visualization types, the interactions (if any) used in these visualizations, and their corresponding code implementations.

We find that standard visualizations (Bar charts, Geographic maps, Line Charts, Scatterplots and Graphs) account for over 80% of the visualizations in our corpus, consistent with prior work [1]. Furthermore, we observe a consistent pattern across our corpus: users' implementation strategies were largely the same within each visualization type. We also find that interactions feature prominently within users' D3 implementations, and certain interaction types are commonly associated with specific visualization types. However a notable fraction of users also created *bespoke* visualizations (see Figure 1), suggesting quantifiable limits to common design patterns.

To make our findings actionable, we synthesize visualization templates for eight popular D3 visualizations, which encapsulate the observed implementation practices of D3 users. These templates are available through our open source repository [1]. Informed by our observations, we discuss critical design considerations for future visualization programming tools that involve code reuse and user-driven recommendations for visualization and interaction design.

To summarize, we make the following contributions in this paper:

- We **analyze common visualization and implementation patterns** across 2500 D3 examples from three online repositories.
- We provide a suite of **general purpose templates** encapsulating common implementation strategies for the most popular D3 visualization and interaction types.
- We **discuss critical design considerations** for future visualization programming tools based on our findings.

[1] https://osf.io/k58bp/?view_only=72fa3798bbaa4263b5ad662b26a70cb3.

2 Related Work

Our work builds on research in visualization authoring, interaction taxonomies, and templates. We review the relevant literature below.

Visualization Authoring Tools: There are a plethora of tools for producing visualization designs, including language- (e.g [5, 16, 26, 28]) and GUI-based (e.g [7, 20, 29, 33]) tools. Specification languages are expressive, but require considerable programming skills to use them well [8,15,30]. GUI-based tools eliminate the need for programming, but in return take control away from the user [14] and still have a considerable learning curve [25]. Visualization users may be better supported by focusing on how to simplify the process of using visualization languages. To this end, we investigate users' implementation practices when using languages like D3.

Visualization and Interaction Taxonomies: Interactions are critical because they allow users to elicit a deeper understanding of their data [9]. Heer and Shneiderman provide a taxonomy of 12 interaction tasks grouped into three high-level categories: (1) Data & View Specification tasks, (2) View Manipulation tasks and (3) Process & Provenance tasks [12]. Brehmer and Munzner [6] bridge the gap between high-level and low-level interaction task[s] classification by connecting complex high-level tasks as a sequence of simpler low-level tasks organized by the intent for the task being performed. To model the space of possible interactions, we incorporate tasks from Brehmer and Munzner's typology [6] to analyze the space of interactions used in D3 visualizations.

Templates: Code reuse has been a strongly advocated and documented programming practice [18, 19]. Within the visualization community, some tools utilize templates to generate visualizations [17, 21, 22] and style templates [11]. Our work extends this line of research on using synthesized templates to support code reuse. However, unlike past work that focuses on the underlying style structure of D3 charts [11], our focus is on understanding what users consider to be *intuitive implementation strategies* for D3 and extracting the commonalities across these strategies.

3 Building a Corpus of D3 Examples

To simplify visualization implementation in D3, we first need to gain a deeper understanding of how users currently use D3. The first step of our analysis was to collect a corpus of publicly available D3 examples from the internet. We adopted Battle et al.'s approach of identifying "islands" or specific websites with thousands of D3 examples [1]. We considered three sources, which together represent the most well-known online repositories containing D3 examples: Bl.ocks.org [3], Observable [4] and GitHub [24]. To assess the quality and viability of each dataset for our analysis, we scraped 500 random D3 examples from each corpus and qualitatively analyzed them. To maintain quality within our dataset, a visualization or interaction is only classified if (1) the example explicitly imports D3 **and** (2) when run, the code produces a visualization in the browser.

We found that the GitHub examples were low quality, due primarily to incomplete code. In addition, unlike the (now deprecated) Bl.ocks.org repository, exported Observable examples are designed to operate within the Observable environment rather than standard web projects, making it more difficult to programmatically analyze them for D3 API usage. Overall, the Bl.ocks.org examples were of consistently higher quality than the GitHub and Observable examples[2]. We believe this difference in quality stems from the maturity of the Bl.ocks.org repository compared to Observable, as well as its intended use as a robust repository of D3 examples compared to GitHub. For these reasons, we analyzed an additional 1000 examples from Bl.ocks.org, resulting in a final corpus with 2500 examples. We report on the visualizations found in all three sources but focus the bulk of our analysis in section 6 on the examples from Bl.ocks.org.

[2]The coded examples are available on OSF: `https://osf.io/k58bp/?view_only=72fa3798bbaa4263b5ad662b26a70cb3`.

4 What visualization types do D3 users implement?

We qualitatively coded the corpus by classifying what visualization(s) were implemented in each example according the taxonomy of D3 visualizations observed by Battle et al [1]. This resulted in 21 visualization types which we refer to as *standard* visualizations. Each distinct visualization found in an example was classified separately. For instance, if both a bar chart and scatterplot were used in a multiple linked view visualization, we classified them separately by visualization type. We discuss the results of our analysis below.

Bl.ocks.org Corpus: We coded a total of 1500 D3 examples from Bl.ocks.org, from which 1265 viable visualizations were found. 994 (78.6%) of these visualizations were *standard* D3 visualization types such as scatter plots, area charts, voronoi diagrams, etc.. Bar charts (19.8% n=251), Geographic maps (15.2% n=192), and line charts (10.8% n=137) were most prevalent (see Figure 2a). The top 5 standard visualizations i.e., Bar Charts, Geographic Maps, Line Charts, Scatterplots and Graphs account for 80.1% of all the standard visualizations implemented with the 16 other visualization types [e.g. Heatmap(1.0%), Voronoi(0.9%), Sankey(0.8%) etc.] accounting for the remaining 19.9%, as shown in Figure 2a. We find that 21.4% (n=271) of these visualizations could not be classified by of the 21 standard visualization types. These examples were diagrams, art, or highly specialized visualizations, such as a departures board for flights, braille clocks, etc., which we classify as *bespoke* visualizations (see Figure 1 for examples).

Our analysis shows that the overwhelming majority of visualizations still conform to the most common visualization types observed on the web. Consistent with prior work [1], just 5 visualization types account for the vast majority of these common visualizations. This suggests that we can support the needs of most D3 users by *focusing on the most popular visualizations from our dataset*.

Observable Corpus: We coded 500 Observable notebooks. However, 49.2% of these notebooks were unrelated to D3. For example, they did not involve data visualizations or used other visualization tools [3]. Of the remaining 254 notebooks, we found 304 viable visualizations, of which 237 (78%) were standard D3 visualization types comprised of Bar charts (20.7% n=63), Line charts (13.2%, n=40), and Scatterplots (12.2%, n=37). The distribution of the Observable visualizations mimicks that of the Bl.ocks.org corpus as the top 5 visualizations account for 79% of the observed visualizations while 11 visualization types account for the remaining 21% of the visualizations. However the rankings for the top 5 visualization types are different in both corpora as seen in Figure 2a.

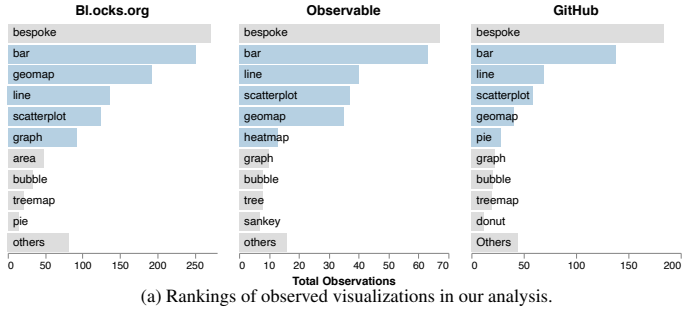
We observed a total of 67 (22%) *bespoke* visualizations in our corpus. These visualizations range from Hyperboloid plots to Cartograms as seen in Figure 1. We find that the bespoke charts created in Observable were more complicated than those available in other mediums. This may be a result of newer visualization libraries and extensions that are available in the Observable environment.

GitHub Corpus: 500 GitHub repositories were coded in our analysis. While all of these repositories imported the D3 API, only 366 contained valid data visualizations. We found 638 visualizations of which 71.2% (n=454) were standard D3 visualization types. The most popular visualization types were Bar charts (30.4% n=138), Line charts (15.2% n=69) and Scatterplots (13% n=59). Similar to our other corpora, the top 5 visualizations make up 73.8% of all standard visualizations on the internet as seen in Figure 2a. We observed a total of 184 (28.8%) *bespoke* visualizations in our corpus.

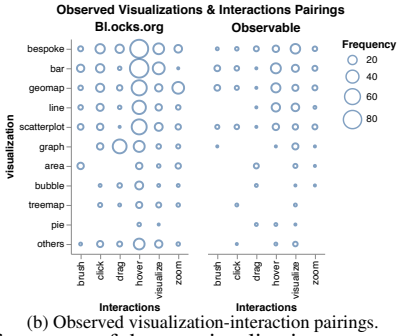
5 What interaction types do D3 users implement?

Next, we classify the kinds of interactions users implement in D3, using a similar approach to section 4. However, existing interaction taxonomies focus on interface elements rather than code components,

[3]Please see `https://observablehq.com/@thisistaimur/warc-study-analysis` and `https://observablehq.com/@aldo/tuftes-charts-in-vega-lite` for examples.



(a) Rankings of observed visualizations in our analysis.



(b) Observed visualization-interaction pairings.

Figure 2: Results from analysis carried out on the three example corpus (a) presents the frequency of the top visualization types and their corresponding interactions observed in our datasets. Visualizations with 1% and less frequency are captured in the *"others"* category. (b) shows the number of times different interaction types were implemented for each visualization type in our corpus

requiring us to translate D3 API components into abstract interaction types. In examining the D3 API and documentation, we extracted 6 common interaction widget types: Brush, Click, Drag, Hover, Visualize, and Zoom. We classify these interactions using Brehmer and Munzer's multi-level typology of visualization tasks [6].

- **Select:** highlighting data points to emphasize salient information, often using "*Brush*", "*Hover*", and "*Click*" widgets.
- **Encode:** changing which data attributes are encoded in a visualization, often using a GUI widget; we use the term *"Visualize"* to represent all such encoding interactions.
- **Navigate:** Re-centering a visualisation on a specific subset of points and granularity, often using "*Zoom & Pan*" widget.
- **Arrange:** sorting and organizing marks within the visualization, such as by using a "*Drag*" widget.

In general we observe a high number of interactions being implemented within visualizations suggesting that users may follow recommended best practices to enable interaction when programming their visualizations [31]. We discuss the details of our analysis for the Bl.ocks.org and Observable corpus below [4].

Bl.ocks.org Corpus Of the 1265 viable examples coded, 659 (52.1%) contained interactions. 859 interaction implementations were identified within these examples, with the most popular widget types being Hover (n=390), Visualize (n=118) and Click (n=100). We observed 62.6% of all interactions were used to "*Brush*", "*Hover*", and "*Click*" specific data points in visualizations. Mapping these observations to the Brehmer and Munzer typology, we find **selection** to be the main form of interaction used, being implemented in 62.6% of D3 visualizations in our analysis. 13.7% of implemented interactions **encode** new data attributes, 13.2 % support **navigation** through visualizations and 10.5% to **arrange** marks on the screen using the *"Drag"* widget. Our findings also revealed that certain interaction types were often implemented *together* in the examples. Examining these associations, we identify 39 distinct pairs of interactions with *"Click and Hover"* being the most frequent pairing representing 14% of all occurrences.

Observable Corpus We had a total of 304 visualization examples in our Observable corpus, of which 48.2% were interactive. 42% of all interactions represented **selections** and a total of 174 interactions were identified with **encoding** new data attributes using the *Visualize* widget being used 35.6% (n=62) of the time. 12.1% with **navigating** through visualizations, and 10.3% with **arranging** visual elements in visualizations. 58 of these interactions (33.3%) were implemented in pairs with 12.1% belonging to the *"Visualize and Hover"* pairs.

6 How are these visualizations being Implemented?

A key step in clarifying the needs of D3 users is to understand *how* they implement visualizations. For each visualization and interaction type, we examined multiple user implementations to understand: the syntactical correctness of each example, the organization and structure of the code, and the API calls used.

6.1 Code Structure

At the low level, users' programs vary slightly for visualizations of the same type, such as by variable names, white space used, and so on. However, when the code is examined at the structural level, the overall order and API calls remain the same. For example, the default code order across visualizations begins with defining the dimensions for the visualization (i.e. width, height, and margins) and creating an SVG object where subsequent visualization elements will be housed. The baseline code for specific visualization types follow almost the same structure. For example, the highlighted code blocks in Figure 3a and Figure 3b highlight similar code structures for defining point marks in two different scatterplot implementations, with slight differences to fit users' datasets. Several examples even contained direct links to other repositories as inspiration. Oftentimes, multiple lines of code were copied directly from these reference repositories. This corroborates past observations on code copying as a common implementation strategy in the D3 community [2, 13].

6.2 Use of Interactions across Visualization Types

To expand upon our observation, we measure how often popular interaction widgets are implemented for each visualization type.

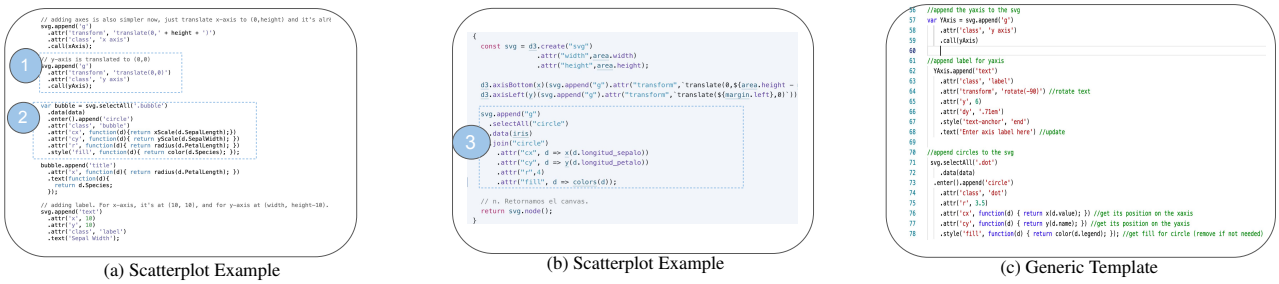
Some interaction widgets appear to be universal. We found that certain widget types were pervasive across most visualizations. For example, hovering is the most common interaction widget for nearly every visualization type observed in Figure 2b. These findings suggest that certain widget types may be considered universal, hinting at code blocks that may be strong candidates for reuse.

The number of interaction widgets varies across visualization types. For certain visualization types (e.g. Graphs) including at least one widget was the convention. For example, 49% of the line chart examples implemented had no interaction widgets, whereas 82% of the graph examples had at least one widget implemented.

Users implement interaction widgets according to visualization type. We find that the types of interaction widgets implemented varies by visualization type (see Figure 2b). For example, we see in Figure 2 that line charts have a wider set of implemented interaction widgets compared to area charts, where hovering was more prevalent in line charts and brushing in area charts. We also see that geographic maps are more likely to have zooming implemented compared to other visualization types, and graphs are more likely to support dragging. Furthermore, the set of interactions implemented were dependent on the visualization type. For example, brushing and hovering were often implemented together in scatterplots.

These findings indicate that for certain D3 implementations, users could benefit from multiple modes of interaction to explore a visualization. For example, the user may (for graphs) or may not (for bar charts) need to consider incorporating interactions and may want to use specific sets of interactions to match existing D3 examples.

[4] Interactions were not coded for GitHub due to data quality issues.



(a) Scatterplot Example (b) Scatterplot Example (c) Generic Template

Figure 3: Reoccurring code blocks (1, 2 & 3) in two scatterplot examples (A & B) are synthesized into a generic scatterplots template (C).

6.3 Generating Reusable D3 Templates

Given an understanding of the most common visualizations (section 4) and interactions (section 5) implemented in D3, we sought to ease the burden of implementing these visualizations. To do this, we randomly selected examples of eight common visualization types and translated common implementation patterns (subsection 6.1) into general-purpose code templates. While publicly available D3 templates exist on platforms like D3Live [17], they are curated based on the template designer's understanding of D3. Our user-driven template synthesis approach encapsulates common programming practices from *hundreds* of D3 users into a single template.

D3 API calls used within examples remain consistent for visualization types. For example, scatterplots and bubble charts consistently contain calls to create circle marks. We rely on D3's API structure and our observations of common D3 code structures to infer the purpose of each line of code and the role they play in creating a visualization. Through this analysis, we manually extracted common code sections for each visualization type. Using scatterplots as an example, we identified common code sections for creating circle marks as shown in Figure 3a & Figure 3b. These code sections were combined to generate a working generic implementation of each visualization that can be modified to fit a user's dataset (see Figure 3c). These generic implementations can be used as templates [5].

7 Discussion

Using our insights into how D3 users implement visualizations, previous work, and our own experiences working with D3, we provide design considerations for simplifying the visualization programming process with visualization languages.

C1: Maximize code component reuse: Code reuse is a documented and advocated skill within the programming and D3 community [2, 13]. Our analysis in section 6 reveals striking structural similarities for various visualization types. Users tend to implement D3 visualizations in consistent ways, where these practices are largely drawn from existing examples. However, recent literature also finds that users often struggle to reuse existing D3 examples because they are typically *not modularized*, making it hard to extract only the relevant functionality [2]. We could speed up the implementation process by automating the adoption of established coding conventions across visualization and interaction types. Based on this premise, we synthesize code *templates* from existing D3 examples to capture existing implementation patterns among D3 users, which could be adopted in future visualization tools. Furthermore, we annotate major code blocks within each template with their intended purpose. This makes it easy for users to quickly associate a code segment in the flow of a template with the specific task it performs.

C2: Use usage statistics to drive adaptive recommendations: Our analysis in section 6 shows that a high fraction of D3 examples allow their end users to explore the underlying data through interactions, reinforcing the importance of interaction in visualizations [9]. In contrast, we see an emphasis on recommending visualization designs but not interaction designs in most visualization recommendation and automated design systems [32]. We observed many combinations of visualizations and interactions. As a result, D3 users interested in implementing interactions would have to sift through numerous examples to identify the appropriate implementation conventions for their target visualization type. To save time and effort, we can instead automatically recommend a curated list of appropriate interactions to implement for a given visualization type.

C3: Involve the community in tool refinement: The large and active D3 community provides a unique opportunity to *co-design* effective visualization tools. For example, recommendation models could be improved by combining visualization heuristics and reinforcement learning to actively learn from the behaviour of D3 users. In terms of interactions, recommenders could infer the current visualization context from the user's code and and compare it with similar visualizations created in the past to provide customized recommendations for potential interactions to implement. Community members could even take a more active role in the tuning of recommenders by providing direct guidance, e.g., annotations, for what the underlying models *should learn* (e.g., code modularization, perceptually effective encoding parameters) or *should not learn* (e.g., poor coding conventions or encoding decisions) from existing examples.

7.1 Limitations

Our work relies on the efforts of thousands of D3 users who publish their work on D3 forums online. Our results are representative of the this population of users but may not generalize to other contexts of D3 usage such as those who use D3 to implement visualizations for news articles or internal usage within organizations. Furthermore, our analysis of the interactions in D3 focuses mainly on interactions that are supported by the native D3 API. We acknowledge that this potentially excludes other interactions that are supported by external modules and libraries, thereby limiting the richness of interaction types found in our study. However, our analysis techniques can easily be applied to other languages, modules and libraries for interactive visualization programming (e.g., Vega-Lite [27]). We also encourage the community to conduct studies in the future to verify how and whether templates simplify the visualization design process.

8 Conclusion and Future Work

In this paper, we investigate visual and interactive design patterns implemented by D3 users through 2500 examples shared in three online repositories (Bl.ocks.org, Observable, GitHub). Five visualization types accounted for over 80% of all the D3 visualizations we observed. We observed that users generally used the same strategies to implement the same visualization type. Furthermore, certain interaction types were frequently paired with specific visualization types. We extracted the common implementation strategies we observed and synthesized templates representing universal building blocks for implementing the most popular D3 visualizations and interactions. These templates provide a baseline that D3 users can build upon to program new visualizations with less time and effort.

Acknowledgments

This work was supported in part by NSF award IIS-1850115 and a VMWare Early Career Faculty Grant.

[5] The curated templates are available on OSF: `https://osf.io/k58bp/?view_only=72fa3798bbaa4263b5ad662b26a70cb3`

REFERENCES

[1] L. Battle, P. Duan, Z. Miranda, D. Mukusheva, R. Chang, and M. Stonebraker. Beagle: Automated extraction and interpretation of visualizations from the web. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–8, 2018.

[2] L. Battle, D. Feng, and K. Webber. Exploring d3 implementation challenges on stack overflow. In *2022 IEEE Visualization Conference (VIS)*. IEEE, 2022.

[3] M. Bostock. Popular blocks, 2016 (accessed November, 2019).

[4] M. Bostock and M. Meckfessel. Observable, 2016 (accessed July, 2021).

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[6] M. Brehmer and T. Munzner. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, 2013.

[7] H. Carr, P. Rheingans, H. Schumann, A. Satyanarayan, and J. Heer. Lyra: An interactive visualization design environment. In *Eurographics Conference on Visualization*, vol. 33, p. 10. Citeseer, 2014.

[8] V. Dibia and Ç. Demiralp. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications*, 39(5):33–46, 2019.

[9] E. Dimara and C. Perin. What is interaction for data visualization? *IEEE Transactions on Visualization and Computer Graphics*, 26(1):119–129, 2020.

[10] J. Harper and M. Agrawala. Deconstructing and restyling d3 visualizations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, p. 253–262. Association for Computing Machinery, New York, NY, USA, 2014. doi: 10.1145/2642918.2647411

[11] J. Harper and M. Agrawala. Converting basic d3 charts into reusable style templates. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1274–1286, 2018. doi: 10.1109/TVCG.2017.2659744

[12] J. Heer and B. Shneiderman. Interactive dynamics for visual analysis. *Commun. ACM*, 55(4):45–54, Apr. 2012. doi: 10.1145/2133806.2133821

[13] E. Hoque and M. Agrawala. Searching the visual style and structure of d3 visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1236–1245, 2020. doi: 10.1109/TVCG.2019.2934431

[14] E. Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 159–166, 1999.

[15] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2019.

[16] P. T. Inc. Collaborative data science, 2015.

[17] E. Katzenstein. D3.js/live, 2016 (accessed August, 2021).

[18] M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in oopl. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04.*, pp. 83–92. IEEE, 2004.

[19] Y. Lin, G. Meng, Y. Xue, Z. Xing, J. Sun, X. Peng, Y. Liu, W. Zhao, and J. Dong. Mining implicit design templates for actionable code reuse. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 394–404. IEEE, 2017.

[20] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, p. 1–13. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3173574.3173697

[21] M. Mauri, T. Elli, G. Caviglia, G. Uboldi, and M. Azzi. Rawgraphs: A visualisation platform to create open outputs. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter*, CHItaly '17. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3125571.3125585

[22] A. M. McNutt and R. Chugh. *Integrated Visualization Editing via Parameterized Declarative Templates*. Association for Computing Machinery, New York, NY, USA, 2021.

[23] H. Mei, Y. Ma, Y. Wei, and W. Chen. The design space of construction tools for information visualization: A survey. *Journal of Visual Languages & Computing*, 44:120–132, 2018.

[24] T. Preston-Werner, C. Wanstrath, P. Hyett, and S. Chaconl. Github, 2008 (accessed July, 2019).

[25] A. Satyanarayan, B. Lee, D. Ren, J. Heer, J. Stasko, J. Thompson, M. Brehmer, and Z. Liu. Critical reflections on visualization authoring systems. *IEEE transactions on visualization and computer graphics*, 26(1):461–471, 2019.

[26] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vegalite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.

[27] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vegalite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017.

[28] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, 2016.

[29] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[30] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. Parameswaran. Towards visualization recommendation systems. *ACM SIGMOD Record*, 45(4):34–39, 2017.

[31] J. S. Yi, Y. a. Kang, J. Stasko, and J. A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, 2007.

[32] Z. Zeng, P. Moh, F. Du, J. Hoffswell, T. Y. Lee, S. Malik, E. Koh, and L. Battle. An evaluation-focused framework for visualization recommendation algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):346–356, 2022. doi: 10.1109/TVCG.2021.3114814

[33] J. Zong, D. Barnwal, R. Neogy, and A. Satyanarayan. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Transactions on Visualization and Computer Graphics*, 2020.