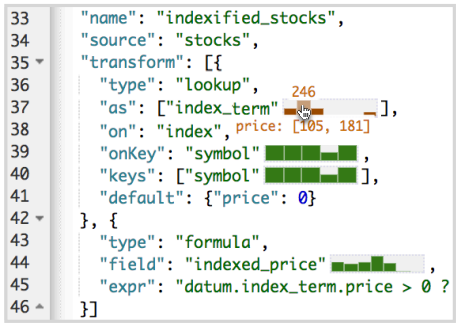

Figure 1: A code snippet with in situ visualizations of program variables in Vega: a declarative visualization grammar. Histograms show the distribution of values for array variables, with the count and range shown on hover. The `symbol` variable is an array of five unique strings representing different companies (AAPL, AMZN, IBM, GOOG, and MSFT), one of which occurs less frequently in the dataset than the others (GOOG). The `indexed_price` variable is an array of numbers corresponding to the stock price. Whereas the `symbol` and `indexed_price` variables are both arrays of a simple type, the `index_term` variable is an array of objects; the histogram is colored orange to differentiate it from the others and shows only the value distribution for the `index_term.price` property.

Languages & Visualizations to Enable Effective End User Programming

Jane Hoffswell
University of Washington
Seattle, WA, USA
jhoffs@cs.washington.edu

ABSTRACT

Programming requires expertise to employ effectively. My research aims to help end user programmers more effectively *author*, *understand*, and *reuse* code and data through the design of new languages and program visualization tools. New programming languages can raise the level of abstraction to focus on relevant domain-specific details. Improved tools can better align with and enrich end user programmers' mental models. Visualizing program state and behavior promotes program understanding, and can proactively surface surprising or incorrect results. My future work proposes to explore new visualization techniques and languages to facilitate understanding of constraint programming systems.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; **Visualization systems and tools**; *Graph drawings*; • **Software and its engineering** → *Constraints*.

KEYWORDS

End user programming; program understanding; debugging; visualization; graph layout; constraints.

ACM Reference Format:
Jane Hoffswell. 2019. Languages & Visualizations to Enable Effective End User Programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI'19 Extended Abstracts), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3290607.3299067

CHI'19 Extended Abstracts, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5971-9/19/05.
https://doi.org/10.1145/3290607.3299067

Thesis Statement: The design of new languages and program visualization tools that raise the level of abstraction from low-level system details to domain-specific concepts and operations can help end user programmers better *author*, *understand*, and *reuse* code and data.

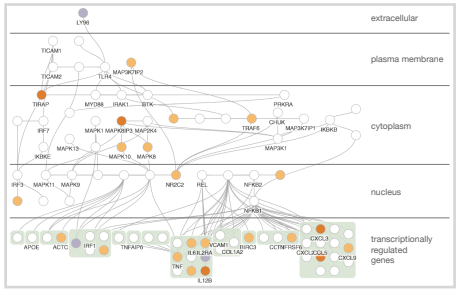

Figure 2: The layout for the TLR4 biological system produced using only eight constraints in SetCoLa: a language for high-level, domain-specific graph layout constraints. Layers correspond to the location of biomolecules in a cell; immune response outcomes are grouped at the bottom by molecular function.

INTRODUCTION

Common programming paradigms require expertise to employ effectively, making them inaccessible to a wide group of end user programmers. End user programmers often have unique expertise that informs the types of computation and development tasks they need to perform [7]. My work aims to help end user programmers more effectively *author*, *understand*, and *reuse* code through the design of languages and visualization tools that shift the focus from low-level system details to important domain-specific concepts and operations. My research into holistic code visualizations [5, 6] aims to **support program understanding during all phases of the development process**, not just while debugging. I further explore the design of languages that allow end user programmers to develop **customized visualizations using their unique domain-specific knowledge** [4, 12]. My work on SetCoLa [4] demonstrates the expressive power of constraint programming systems, but does not address the program understanding needs explored in my prior work. Going forward, my dissertation aims to **facilitate program understanding of constraint programming systems** for end user programmers, by leveraging the lessons learned from my research into holistic code visualizations.

Background: I am a fifth year PhD student working with Professor Jeffrey Heer in the Paul G. Allen School of Computer Science and Engineering at the University of Washington. I conduct research as part of the Interactive Data Lab, with a focus on the design of new interactive systems for facilitating code authoring and program understanding through the use of visualization. I am currently refining the scope of my thesis on the design of end user programming systems for constraint programming.

RELATED WORK

This section covers only a small subset of the related work that has inspired my research. End user programming engages a wide group of potential users that incorporate domain-specific knowledge into their computational tasks [7]. Ko et al. describe the types of programming tasks commonly performed [9] and program understanding challenges [8]. Bret Victor presents design considerations for systems that leverage visualizations of program behavior [13]. Constraint systems are a flexible approach for domain-specific applications such as graph layout [3], user interface layout [1], and recently the compilation and reapplication of visualization design guidelines [10]. These topics inspire my research on new languages and program visualization tools to facilitate end user programming.

RESEARCH PROGRESS

My prior work explores the design of languages that allow users to focus on domain-specific concepts and operations rather than low-level system details [4, 12] and the design of visualization tools to support program understanding [5, 6]. Figure 4 shows the projected timeline for my dissertation.

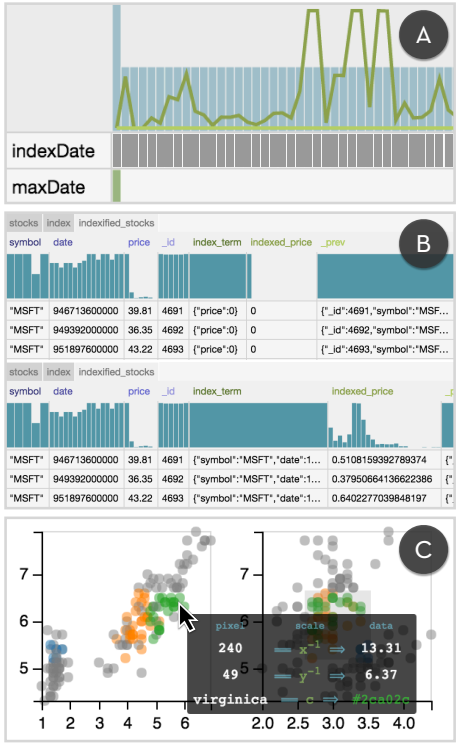

Figure 3: Several visualizations that facilitate program understanding in Vega. (a) A timeline shows all modifications to a variable (gray boxes for `indexDate`). The green line shows the variability of the `indexed_price` variable from the underlying data source. (b) When the variability of the `indexed_price` peaks, the property contains all zeroes (top), rather than a distribution of values (bottom). (c) The user inspects the programmatic transformations used in the code as a tooltip directly on the output Vega visualization.

Authoring & Reusing Domain-Specific Graph Layouts

Domain experts often create complex visualizations of graph data based on domain-specific properties relevant to the layout. For example, customized graph layouts are commonly used to visualize biological systems based on knowledge of cellular structure (Figure 2). However, common approaches to domain-specific layout require domain experts to either use ill-fitting techniques that are not ideal for the task at hand, or to create tools specifically designed for their particular use case (e.g., Cerebral [2]). Constraints support flexible custom layouts, but can be tedious to author when large layouts require hundreds of constraints between individual nodes in the original graph (e.g., WebCoLa [3]).

To facilitate authoring custom graph layouts, I developed SetCoLa [4]: a language for specifying high-level constraints for graph layout based on domain-specific properties of the graph. Whereas prior constraint systems utilize node-level constraints between specific nodes [3], SetCoLa **reduces the number of user-authored constraints by one to two orders of magnitude.** In SetCoLa, constraints are applied to sets of nodes based on the domain-specific properties of the nodes rather than between pairs of nodes in the graph. The layout in Figure 2 requires only eight constraints in SetCoLa; the core SetCoLa layout requires just one constraint to specify the order of the layers and two constraints to specify the boundaries of the graph, with the remaining constraints customizing the groupings and inter-node padding. This specification generates 363 constraints for the underlying constraint engine, WebCoLa [3]. This approach **facilitates program understanding** because there is a direct mapping between the graph layout and domain-specific properties of the nodes. This specification strategy also **supports reuse of custom graph layouts** because the layout only leverages domain-specific properties, which can be generalized to other graphs with the same properties.

Visualizations to Aid Program Understanding & Authoring

My prior work also explores the design of Vega [12]: a declarative visualization grammar that allows programmers to focus on the design of an interactive visualization rather than the low-level implementation details. However, this approach introduces a gap between the code the programmer writes and the system output, which often requires a complex mental model of the behavior to understand and debug. To help end user programmers better *author* and *understand* Vega code, I designed several approaches to program visualization [5, 6]. These visualizations provide a holistic view of the program behavior by *automatically* visualizing the state and/or history for all of the variables in a Vega program.

The system automatically displays all value updates to all program variables on a timeline (Figure 3a). The underlying data for the program is displayed in a separate tab, with histograms showing the distribution of the values for each data property (Figure 3b); the variability of these data distributions are visualized above the timeline (e.g., the green line in Figure 3a). End user programmers can use time-traveling debugging to revisit past runtime states by selecting different points on the timeline. A tooltip

Figure 4: Dissertation Timeline

displays programmatic transformations to the data directly on the output Vega visualization (Figure 3c). In an evaluation with end user programmers unfamiliar with both the specific code and the Vega programming language, the participants could **effectively understand the source code to identify bugs or crucial dependencies** in the underlying data flow behavior.

However, this debugging environment required end user programmers to intentionally switch between different views based on their current programming task (e.g., reviewing code, inspecting the timeline, or viewing the data). In my follow-up work [6], program visualizations are displayed directly inline in the source code (Figure 1), thus allowing programmers to inspect the history of values for program variables while unloading the burden of recalling or mentally tracking the program execution. Instead of switching between multiple views to write, test, and debug their code, programmers can **maintain a focus on authoring new code** while viewing the behavior of program variables inline.

FUTURE WORK

In future work, I propose to explore new techniques to encourage an effective exchange between end user programmers and the technology they use. I am particularly interested in how new visualization techniques can be used alongside automatic program analysis to recommend or attract attention to potential areas of interest in a programmer's code. Such approaches should help end user programmers focus their attention during debugging or program understanding tasks to reduce the amount of wasted development time [11]. Furthermore, these approaches should be complimentary to the goals of the end user programmer and should not detract from the code authoring process.

I believe that better supporting program understanding in complex domains can enable more end user programmers to engage with systems that have otherwise required expertise to use effectively. One such area that I would like to explore is the utility and interpretation of constraints. Constraints are a flexible way to express behaviors or circumstances relevant to a user; for example, constraints can be used for graph or interface layout, scheduling, and various forms of optimization or prioritization. However, the execution and invalidation of constraints can be hard to comprehend in complex systems. By developing new approaches to program understanding, I hope to open the domain to a wider group of potential users and better support the effective use of new constraint-based approaches.

CONTRIBUTIONS & CONCLUSION

My work on new domain-specific languages [4, 12] and automatic visualization tools [5, 6] helps end user programmers better navigate and understand code by allowing them to focus on the domain-specific concepts and operations of interest rather than the low-level system details. My work on SetCoLa [4] demonstrates the utility of high-level languages for customized graph layouts and the potential benefits of constraint-based systems. My future work aims to combine constraint programming with new visualization techniques to facilitate code *authoring*, *understanding*, and *reuse*.

ACKNOWLEDGMENTS

I would like to thank my collaborators, the Interactive Data Lab at the University of Washington, and the many others who have supported me in my research thus far. Special thanks to my advisor, Jeffrey Heer. This work was supported by a Moore Foundation Data-Driven Discovery Investigator Award and the National Science Foundation (IIS-1758030).

REFERENCES

[1] Greg J Badros, Alan Borning, and Peter J Stuckey. 2001. The Cassowary Linear Arithmetic Constraint Solving Algorithm. *ACM Transactions on Computer-Human Interaction (TOCHI)* (2001). https://doi.org/10.1145/504704.504705

[2] Aaron Barsky, Tamara Munzner, Jennifer Gardy, and Robert Kincaid. 2008. Cerebral: Visualizing Multiple Experimental Conditions on a Graph with Biological Context. *IEEE Transactions on Visualization & Computer Graphics* (2008). https://doi.org/10.1109/TVCG.2008.117

[3] Tim Dwyer. 2017. cola.js: Constraint-Based Layout in the Browser. http://marvl.infotech.monash.edu/webcola/. Accessed: 2018-10-10.

[4] Jane Hoffswell, Alan Borning, and Jeffrey Heer. 2018. SetCoLa: High-Level Constraints for Graph Layout. *Computer Graphics Forum (Proc. EuroVis)* (2018). https://doi.org/10.1111/cgf.13440

[5] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2016. Visual Debugging Techniques for Reactive Data Visualization. *Computer Graphics Forum (Proc. EuroVis)* (2016). https://doi.org/10.1111/cgf.12903

[6] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2018. Augmenting Code with In Situ Visualizations to Aid Program Understanding. *ACM Human Factors in Computing Systems (CHI)* (2018). https://doi.org/10.1145/3173574.3174106

[7] Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. 2011. The State of the Art in End-User Software Engineering. *ACM Computing Surveys (CSUR)* (2011). https://doi.org/10.1145/1922649.1922658

[8] Andrew J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. In *IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 199–206. https://doi.org/10.1109/VLHCC.2004.47

[9] Andrew J Ko, Brad A Myers, Michael J Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information During Software Maintenance Tasks. *IEEE Transactions on Software Engineering* (2006). https://doi.org/10.1109/TSE.2006.116

[10] Dominik Moritz, Chenglong Wang, Gregory Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. 2019. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Transactions on Visualization & Computer Graphics (Proc. InfoVis)* (2019). https://doi.org/10.1109/TVCG.2018.2865240

[11] David Saff and Michael D Ernst. 2003. Reducing Wasted Development Time Via Continuous Testing. In *14th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 281–292. https://doi.org/10.1109/ISSRE.2003.1251050

[12] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2015. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization & Computer Graphics (Proc. InfoVis)* (2015). https://doi.org/10.1109/TVCG.2015.2467091

[13] Bret Victor. 2012. Learnable Programming: Designing a Programming System for Understanding Programs. http://worrydream.com/LearnableProgramming. Accessed: 2018-26-11.