Instructions

For each reading assignment, do the following:

1. **Read the required reading in full**. There will usually be 1 required reading, which should take around a 1 hour to read deeply and in full. In some cases, there will be 2 required readings; however, the overall reading time across all readings should be about the same.

2. **Briefly consult the supplemental readings**. Open up the supplemental readings and orient yourself to them. If the reading is a research paper, understand the abstract and take a look at the figures. If it is a short video, watch it. If the reading is a slide deck, skim through it. The purpose of these readings are not to deeply engage with them, but rather to build up your mental map of what research and inquiry looks like around this topic. These are some of the papers you would want to look at in the future (for your research, or the final project) when you want to understand this topic more deeply.

3. **Readings that are marked as "Reference" do not need to be read**. These readings are listed only if you want to do a deeper dive into the topic on your own.

4. **Submit written commentary**. Write a brief reflection elaborating on what you learned about designing effective programming environments from readings. ***The goal of this written commentary is for you to develop thoughts that will steer your future research and work***. Consider the commentary as a journal entry where you document your evolving understanding of the design of programming environments with the readings as a provocation.

As a rule of thumb, written commentary should be around 300 words long. If you are not sure what to write about, consider touching upon one or more of the following topics:

- What works really well about a tool, and why?
- Does the technology work well in all circumstances (i.e., for all users, for all programming languages)? If not, how does it fall apart in other circumstances?
- What theory and practices from domains that you are intimately familiar with (i.e., from HCI, programming languages, education) suggest better ways to design and evaluate tools like those from the readings?
- Have you tried to use the technology before, or another one like it? If so, what might the readings be missing about the experience of using the technology?
- Are you convinced by the evidence of usability collected to date? If not, what is not convincing, and what studies should be run?
- What are some blindspots in this area of research that need attention?

For days that we speak with an invited speaker, the written commentary will consist of 2 questions that you would like to ask a speaker, each approximately 1 paragraph in length to include sufficient context, and upvoting questions submitted by your peers.

Readings

Unit 1: Inspiration

A dose of inspiration

Victor, Bret. Learnable Programming. http://worrydream.com/LearnableProgramming/.

A feast of demos and critiques

See instructions at https://canvas.upenn.edu/courses/1680358/assignments/10533155.

Unit 2: Literate programming

The present: Jupyter

[**Required**] Kery, Mary Beth, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. "The story in the notebook: Exploratory data science using a literate programming tool." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-11. 2018.
Link

[Supplemental] Rule, Adam, Aurélien Tabard, and James D. Hollan. "Exploration and explanation in computational notebooks." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2018. Link

[Supplemental] Perez, Fernando, and Brian E. Granger. "Project Jupyter: Computational narratives as the engine of collaborative data science." Link

[Supplemental] Wolfram, Stephen. "What is a Computational Essay?" (2017). Link

[Supplemental] I Don't Like Notebooks - Joel Grus - #JupyterCon 2018.
I Don't Like Notebooks - Joel Grus - #JupyterCon 2018 .

[Reference] Lau, Sam, Ian Drosos, Julia M. Markel, and Philip J. Guo. "The design space of computational notebooks: An analysis of 60 systems in academia and industry." In 2020 IEEE

Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 1-11. IEEE, 2020. Link.

The past: WEB

[**Required**] Knuth, Donald Ervin. "Literate programming." The computer journal 27, no. 2 (1984): 97-111. Link.

[**Required**] Knuth, Donald E. "TEX: the Program." Reading: Addison-Wesley (1984). **Instructions**: Read the preface. Then pick **1 chapter** you are interested in and try to get through as much as you can in 20 minutes. Link.

[Supplemental] Ramsey, Norman. "Literate programming simplified." IEEE software 11, no. 5 (1994): 97-105. Link.

[Special event] Distinguished Lecturer: Sorin Lerner

[**Required**] Lerner, Sorin. "Projection boxes: On-the-fly reconfigurable visualization for live programming." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-7. 2020. Link.

[Supplemental] Lerner, Sorin. "Focused Live Programming with Loop Seeds." In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 607-613. 2020. Link.

[Supplemental] Ferdowsifard, Kasra, Allen Ordookhanians, Hila Peleg, Sorin Lerner, and Nadia Polikarpova. "Small-step live programming by example." In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 614-626. 2020. Link.

[Supplemental] Ringer, Talia, Alex Sanchez-Stern, Dan Grossman, and Sorin Lerner. "REPLica: REPL instrumentation for Coq analysis." In Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, pp. 99-113. 2020. Link.

[Reference] Leung, Alan, and Sorin Lerner. "Parsimony: An IDE for example-guided synthesis of lexers and parsers." In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 815-825. IEEE, 2017. Link.

[Reference] Sarracino, John, Odaris Barrios-Arciga, Jasmine Zhu, Noah Marcus, Sorin Lerner, and Ben Wiedermann. "User-guided synthesis of interactive diagrams." In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 195-207. 2017. Link.

[Reference] Foster, Stephen R., Sorin Lerner, and William G. Griswold. "Seamless Integration of Coding and Gameplay: Writing Code Without Knowing it." In FDG. 2015. Link.

The present: Tutorials

[**Required**] Head, Andrew, Jason Jiang, James Smith, Marti A. Hearst, and Björn Hartmann. "Composing flexibly-organized step-by-step tutorials from linked source code, snippets, and outputs." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2020.

[Supplemental] Mysore, Alok, and Philip J. Guo. "Torta: Generating mixed-media gui and command-line app tutorials using operating-system-wide activity tracing." In Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, pp. 703-714. 2017.

[Supplemental] Mirhosseini, Samim, and Chris Parnin. "Docable: evaluating the executability of software tutorials." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 375-385. 2020.

[Supplemental] Khandwala, Kandarp, and Philip J. Guo. "Codemotion: expanding the design space of learner interactions with computer programming tutorial videos." In Proceedings of the Fifth Annual ACM Conference on Learning at Scale, pp. 1-10. 2018.

[Supplemental] Kim, Ada S., and Amy J. Ko. "A pedagogical analysis of online coding tutorials." In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp. 321-326. 2017.

[Reference] Jupyter Book. https://jupyterbook.org/en/stable/intro.html.

A feast of demos and critiques

See instructions at https://canvas.upenn.edu/courses/1680358/assignments/10565830.

Reviewing evidence of usability

Special reading response instructions: This week, I want you to focus your reading responses on what you *wish* we knew about the usability of literate programming tools (either for authors or readers) that studies to date have not yet shown us.

[**Required**] Head, Andrew. Interactive Program Distillation. Ph.D. thesis. **Instructions**: Read p21-43 only. In your reading application, highlight passages that describe evaluations of the proposed tools, and try to synthesize an understanding of what we can generalize on the basis of this prior evidence. Link.

[**Required**] Ramsey, Norman and Carla Marceau. “Literate Programming on a Team Project.” Software—Practice and Experience 21.7 (1991), pp. 677–683. Link.

[*Supplemental*] Chattopadhyay, Souti, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. "What's wrong with computational notebooks? Pain points, needs, and design opportunities." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2020. **Instructions**: pay particular attention to the passages about “Manage Code,” “Archival,” “Share and Collaborate,” “Reproduce and Reuse.” Link.

[Supplemental] Shum, Stephen and Curtis Cook. “Using Literate Programming to Teach Good Programming Practices.” Proceedings of the Technical Symposium on Computer Science Education. ACM, 1994, pp. 66–70. Link.

[Supplemental] Childs, Bart, Deborah Dunn, and William Lively. “Teaching CS/1 Courses in a Literate Manner.” TUGboat 16.3 (1995), p. 8. Link’.

[*Supplemental*] Parnin, Chris, Christoph Treude, and Margaret-Anne Storey. "Blogging developer knowledge: Motivations, challenges, and future directions." In 2013 21st International Conference on Program Comprehension (ICPC), pp. 211-214. IEEE, 2013. Link.

[Reference] Rule, Adam, Aurélien Tabard, and James D. Hollan. "Exploration and explanation in computational notebooks." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2018. Link

[Reference] Thimbleby, H. “Experiences of ‘Literate Programming’ using cweb (a variant of Knuth’s WEB).” The Computer Journal 29.3 (1986), pp. 201–211. Link.

[Reference] DeLine, Robert, and Danyel Fisher. "Supporting exploratory data analysis with live programming." In 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 111-119. IEEE, 2015. Link.

[Reference] Van Wyk, Christopher J. “Literate programming: An assessment.” Communications of the ACM 33.3 (1990), pp. 361–363. Link.

[Reference] Literate Programming in the Large. Start around 7:00.

Unit 3: Live programming

The past: SmallTalk

[**Required**] Kay, Alan C. "The early history of Smalltalk." In History of programming languages---II, pp. 511-598. 1996. Link.

[**Required**] Tanimoto, Steven L. "VIVA: A visual language for image processing." Journal of Visual Languages & Computing 1, no. 2 (1990): 127-139. **Instructions**: Read Sections 1 and 2 only, focusing on the 4 levels of liveness. Link.

[Supplemental] Tanimoto, Steven L. "A perspective on the evolution of live programming." In 2013 1st International Workshop on Live Programming (LIVE), pp. 31-34. IEEE, 2013. Link.

[Reference] Alto System Project: Dan Ingalls demonstrates Smalltalk. Highlights: 10:00, where Dan describes how he got involved in the SmallTalk system. 19:30, where Dan begins demonstrating the system.

[Reference] Squeak. https://squeak.org/. [Squeak is a modern open-source execution environment for SmallTalk].

The present: Live coding demos

[**Required**] Chen, Charles H., and Philip J. Guo. "Improv: Teaching programming at scale via live coding." In Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale, pp. 1-10. 2019. Link.

[**Required**] Guo, Philip. "Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia." In The 34th Annual ACM Symposium on User Interface Software and Technology, pp. 1235-1251. 2021. Link. **Instructions**: Read Sections 3 and 4 only. Link.

[Supplemental] Chen, Yan, Walter S. Lasecki, and Tao Dong. "Towards supporting programming education at scale via live streaming." Proceedings of the ACM on Human-Computer Interaction 4, no. CSCW3 (2021): 1-19. Link.

[Supplemental] Alaboudi, Abdulaziz, and Thomas D. LaToza. "An exploratory study of live-streamed programming." In 2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 5-13. IEEE, 2019. Link.

[Reference] Haaranen, Lassi. "Programming as a performance: Live-streaming and its implications for computer science education." In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 353-358. 2017. Link.

[Reference] Mahoney, Mark. "Storyteller: a tool for creating worked examples." Journal of Computing Sciences in Colleges 34, no. 1 (2018): 137-144. Link.

[Reference] Chi, Peggy, Tao Dong, Christian Frueh, Brian Colonna, Vivek Kwatra, and Irfan Essa. "Synthesis-Assisted Video Prototyping From a Document." In Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology, pp. 1-10. 2022. Link.

[Reference] Chi, Pei-Yu, Sen-Po Hu, and Yang Li. "Doppio: Tracking ui flows and code changes for app development." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-13. 2018. Link.

In-situ visualization

[**Required**] Hoffswell, Jane, Arvind Satyanarayan, and Jeffrey Heer. "Augmenting code with in situ visualizations to aid program understanding." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2018. Link.

[**Required**] Kery, Mary Beth, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. "mage: Fluid moves between code and graphical work in computational notebooks." In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 140-151. 2020. Link. **Instructions**: Read only "Introduction," "Demonstrating Design Space in Tools" and watch this demo video.

[Supplemental] Kang, Hyeonsu and Philip J. Guo. "Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations." Proceedings of the Symposium on User Interface Software and Technology. ACM, 2017, pp. 737–745. Link.

[Supplemental] Omar, Cyrus, Young Seok Yoon, Thomas D. LaToza, and Brad A. Myers. "Active code completion." In 2012 34th International Conference on Software Engineering (ICSE), pp. 859-869. IEEE, 2012. Link.

[Supplemental] Lieber, Tom, Joel R. Brandt, and Rob C. Miller. "Addressing misconceptions about code with always-on programming visualizations." In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2481-2490. 2014. Link.

[Supplemental] Cito, Jürgen, Philipp Leitner, Martin Rinard, and Harald C. Gall. "Interactive production performance feedback in the IDE." In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 971-981. IEEE, 2019. Link.

[Supplemental] Sulír, Matúš, Michaela Bačíková, Sergej Chodarev, and Jaroslav Porubän. "Visual augmentation of source code editors: A systematic mapping study." Journal of Visual Languages & Computing 49 (2018): 46-59. Link.

[Supplemental] "Light Table: the next generation code editor." http://lighttable.com/. **Instructions**: Skim the feature list on the middle of the page.

[Reference] Lerner, Sorin. "Projection boxes: On-the-fly reconfigurable visualization for live programming." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-7. 2020. Link.

Additional readings on live programming

Burckhardt, Sebastian, Manuel Fahndrich, Peli de Halleux, Sean McDirmid, Michal Moskal, Nikolai Tillmann, and Jun Kato. "It's alive! continuous feedback in UI programming." In Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation, pp. 95-104. 2013. Link.

[Special event] Guest Lecturer: Ian Arawjo

[**Required**] Arawjo, Ian. "To write code: The cultural fabrication of programming notation and practice." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-15. 2020. Link. **Instructions**: This paper is in the style of a cultural critique that has not come up in our readings before. While I encourage you to read the paper in full, those sections I would like you to pay particular attention to are “The Culture in Early Programming Notations: Three visions” and “Embracing Heterogeneity in Programming Practice.”

[Supplemental] Arawjo, Ian, Anthony J. DeArmas, Michael Roberts, Shrutarshi Basu, and Tapan Parikh. "Notational Programming for Notebook Environments: A Case Study with Quantum Circuits." (2022). Link.

[Supplemental] Arawjo, Ian, Cheng-Yao Wang, Andrew C. Myers, Erik Andersen, and François Guimbretière. "Teaching programming with gamified semantics." In Proceedings of the 2017 CHI conference on human factors in computing systems, pp. 4911-4923. 2017. Link.

[Reference] Arawjo, Ian, and Ariam Mogos. "Intercultural computing education: Toward justice across difference." ACM Transactions on Computing Education (TOCE) 21, no. 4 (2021): 1-33. Link.

A feast of demos and critiques

See instructions on Canvas (forthcoming).

Unit 4: Design methods

Advice on designing programming languages, Part I

[**Required**] Coblenz, Michael, Gauri Kambhatla, Paulette Koronkevich, Jenna L. Wise, Celeste Barnaby, Joshua Sunshine, Jonathan Aldrich, and Brad A. Myers. "PLIERS: a process that integrates user-centered methods into programming language design." ACM Transactions on Computer-Human Interaction (TOCHI) 28, no. 4 (2021): 1-53. Link.

[Supplemental] Pane, John F., and Brad A. Myers. "Studying the language and structure in non-programmers' solutions to programming problems." International Journal of Human-Computer Studies 54, no. 2 (2001): 237-264. Link.

[Supplemental] Stefik, Andreas, and Stefan Hanenberg. "Methodological irregularities in programming-language research." Computer 50, no. 8 (2017): 60-63. Link.

[Supplemental] Stefik, Andreas, and Susanna Siebert. "An empirical investigation into programming language syntax." ACM Transactions on Computing Education (TOCE) 13, no. 4 (2013): 1-40. Link.

[Supplemental] Karsai, Gabor, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. "Design Guidelines for Domain Specific Languages." Domain-Specific Modeling (DSM'09). Link.

[Reference] Kosar, Tomaž, Marjan Mernik, and Jeffrey C. Carver. "Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments." Empirical software engineering 17, no. 3 (2012): 276-304. Link.

[Reference] Jun, Eunice, Maureen Daum, Jared Roesch, Sarah Chasins, Emery Berger, Rene Just, and Katharina Reinecke. "Tea: A high-level language and runtime system for automating statistical analysis." In Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, pp. 591-603. 2019. Link.

Advice on designing programming languages, Part II

[**Required**] Green, Thomas RG. "Cognitive dimensions of notations." People and computers V (1989): 443-460. Link.

[Supplemental] Boshernitsan, Marat, Susan L. Graham, and Marti A. Hearst. "Aligning development tools with the way programmers think about code changes." In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 567-576. 2007. Link. **Instructions**: Skim as usual, and take a close look at the sections "THEORETICAL FRAMEWORK", "Early Designs", and "Cognitive Dimensions Evaluation".

[Supplemental] Satyanarayan, Arvind, Kanit Wongsuphasawat, and Jeffrey Heer. "Declarative interaction design for data visualization." In Proceedings of the 27th annual ACM symposium on User interface software and technology, pp. 669-678. 2014. Link. **Instructions**: Skim as usual, and take a close look at the section "DISCUSSION: COGNITIVE DIMENSIONS OF NOTATION".

Advice on designing tools

[**Required**] Myers, Brad A., Amy J. Ko, Thomas D. LaToza, and YoungSeok Yoon. "Programmers are users too: Human-centered methods for improving programming tools." Computer 49, no. 7 (2016): 44-52. Link.

[**Required**] Norman, Don. The design of everyday things: Revised and expanded edition. Basic books, 2013. **Instructions**: Skim Chapter 6 on "Design Thinking" (see excerpt in link). Pay particular attention to the sections on "Solving the Correct Problem" and "The Double-Diamond Model of Design". Then, at some later time after this course, find a copy of this book and give the whole book a skim. Link

[Supplemental] Houde, Stephanie, and Charles Hill. "What do prototypes prototype?." In Handbook of human-computer interaction, pp. 367-381. North-Holland, 1997. Link.
[Reference] Beyer, H., & Holtzblatt, K. (1999). Contextual design. interactions, 6(1), 32-42. See Chapter 3: Principles of Contextual Inquiry. Link.

[Reference] Fogarty, James. "Code and contribution in interactive systems research." In Workshop HCITools: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits at CHI. 2017.

[Reference] Wobbrock, Jacob O., and Julie A. Kientz. "Research contributions in human-computer interaction." interactions 23, no. 3 (2016): 38-44.

[Reference] Chasins, Sarah E., Elena L. Glassman, and Joshua Sunshine. "PL and HCI: better together." Communications of the ACM 64, no. 8 (2021): 98-106. Link.

Unit 5: Evaluation methods

How to assess usability

[**Required**] Ledo, David, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. "Evaluation strategies for HCI toolkit research." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-17. 2018. Link.

[Supplemental] Ko, Amy J., Thomas D. LaToza, and Margaret M. Burnett. "A practical guide to controlled experiments of software engineering tools with human participants." Empirical Software Engineering 20, no. 1 (2015): 110-141. Link.

[Supplemental] Olsen Jr, Dan R. "Evaluating user interface systems research." In Proceedings of the 20th annual ACM symposium on User interface software and technology, pp. 251-258. 2007. Link.

[Reference] Nielsen, Jakob. "How to conduct a heuristic evaluation." Nielsen Norman Group 1, no. 1 (1995): 8. Link.

How to assess program comprehension

[**Required**] Pennington, Nancy. "Stimulus structures and mental representations in expert comprehension of computer programs." Cognitive psychology 19, no. 3 (1987): 295-341. **Note**: This paper is a long read; give it a serious effort, particularly in understanding how conclusions were drawn about programmers' mental representations on the basis of data. That said, time box your reading of this paper to about 1 hour. Link.

[Supplemental] Détienne, Françoise. Software design–cognitive aspects. Springer Science & Business Media, 2001. **Instructions**: Skim Chapter 1. Those interested in models of program comprehension should also skim Chapter 3. Link.

[Supplemental] Head, Andrew. Interactive Program Distillation. Ph.D. thesis. **Instructions**: Skim Chapter 2 Section "How do programmers read programs?" and Figure 2.1. Link.

[Supplemental] Crichton, Will, Maneesh Agrawala, and Pat Hanrahan. "The Role of Working Memory in Program Tracing." In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-13. 2021. Link.

[Reference] Ko, Amy J., Thomas D. LaToza, and Margaret M. Burnett. "A practical guide to controlled experiments of software engineering tools with human participants." Empirical Software Engineering 20, no. 1 (2015): 110-141. Link.

[Reference] Boehm-Davis, Deborah A., Robert W. Holt, and Alan C. Schultz. "The role of program structure in software maintenance." International Journal of Man-Machine Studies 36, no. 1 (1992): 21-63. Link.

[Reference] Soloway, Elliot, and Kate Ehrlich. "Empirical studies of programming knowledge." IEEE Transactions on software engineering 5 (1984): 595-609. Link.

[Reference] Crichton, William Perry. "Revisiting Program Slicing with Ownership-based Information Flow." PhD dissertation., Stanford University, 2022. Link. **Instructions**: Read Chapter 2 on "Cognition and Programming."

Unit 6: Advanced live and literate programming techniques

Reviewing evidence of usability of live programming systems

Special instructions: Pick **one** of the following papers, and mark it with your name to "claim" it as your paper to present. Each paper should be read by only 1 reader. Follow these instructions on Canvas on how to reflect on the reading.

1. Wilcox, Eric M., J. William Atwood, Margaret M. Burnett, Jonathan J. Cadiz, and Curtis R. Cook. "Does continuous visual feedback aid debugging in direct-manipulation programming systems?." In Proceedings of the ACM SIGCHI Conference on Human factors in computing systems, pp. 258-265. 1997. Link.

2. Kramer, Jan-Peter, Joachim Kurz, Thorsten Karrer, and Jan Borchers. "How live coding affects developers' coding behavior." In 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 5-8. IEEE, 2014. Link.

3. DeLine, Robert, and Danyel Fisher. "Supporting exploratory data analysis with live programming." In 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 111-119. IEEE, 2015. Link.

4. Hoffswell, Jane, Arvind Satyanarayan, and Jeffrey Heer. "Augmenting code with in situ visualizations to aid program understanding." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2018. Link.

5. Hartmann, Björn, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. "Design as exploration: creating interface alternatives through parallel authoring and runtime tuning." In Proceedings of the 21st annual ACM symposium on User interface software and technology, pp. 91-100. 2008. Link.

6. Omar, Cyrus, Young Seok Yoon, Thomas D. LaToza, and Brad A. Myers. "Active code completion." In 2012 34th International Conference on Software Engineering (ICSE), pp. 859-869. IEEE, 2012. Link.

7. Beck, Fabian, Oliver Moseler, Stephan Diehl, and Günter Daniel Rey. "In situ understanding of performance bottlenecks through visually augmented code." In 2013 21st International Conference on Program Comprehension (ICPC), pp. 63-72. IEEE, 2013. Link.

8. Lieber, Tom, Joel R. Brandt, and Rob C. Miller. "Addressing misconceptions about code with always-on programming visualizations." In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2481-2490. 2014. Link.

9. Kang, Hyeonsu and Philip J. Guo. “Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations.” Proceedings of the Symposium on User Interface Software and Technology. ACM, 2017, pp. 737–745. Link.

10. Zhang, Xiong, and Philip J. Guo. "Ds. js: Turn any webpage into an example-centric live programming environment for learning data science." In Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, pp. 691-702. 2017. Link.

11. Cito, Jürgen, Philipp Leitner, Martin Rinard, and Harald C. Gall. "Interactive production performance feedback in the IDE." In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 971-981. IEEE, 2019. Link.

12. Lerner, Sorin. "Projection boxes: On-the-fly reconfigurable visualization for live programming." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-7. 2020. Link.

13. Lerner, Sorin. "Focused Live Programming with Loop Seeds." In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 607-613. 2020. Link.

14. Ferdowsifard, Kasra, Allen Ordookhanians, Hila Peleg, Sorin Lerner, and Nadia Polikarpova. "Small-step live programming by example." In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 614-626. 2020. Link.

15. DeLine, Robert A. "Glinda: Supporting data science with live programming, GUIs and a Domain-specific Language." In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-11. 2021. Link.

Generating documentation

Note: In anticipation for an in-class activity tomorrow (Wed. 11/9), Hao asks that you bring a code snippet from code you recently wrote consisting of 15 or fewer lines of code. The snippet can be printed out, handwritten, or on your laptop or smartphone. I encourage you to reach out to Hao (what@seas.upenn.edu) with any questions.

[**Required**] Wang, April Yi, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. "Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks." ACM Transactions on Computer-Human Interaction 29, no. 2 (2022): 1-33. Link.

[Supplemental] Li, Yong, Shoaib Kamil, Alec Jacobson, and Yotam Gingold. "I♥ LA: compilable markdown for linear algebra." ACM Transactions on Graphics (TOG) 40, no. 6 (2021): 1-14. Link.

[Supplemental] Head, Andrew, Codanda Appachu, Marti A. Hearst, and Björn Hartmann. "Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code." In 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 3-12. IEEE, 2015. Link.

[Reference] Maalej, Walid, and Martin P. Robillard. "Patterns of knowledge in API reference documentation." IEEE Transactions on Software Engineering 39, no. 9 (2013): 1264-1282. Link.

[Reference] Liu, Zhongxin, Xin Xia, Meng Yan, and Shanping Li. "Automating just-in-time comment updating." In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, pp. 585-597. 2020. Link.

[Reference] Shrestha, Nischal, Titus Barik, and Chris Parnin. "It's like python but: Towards supporting transfer of programming language knowledge." In 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 177-185. IEEE, 2018. Link.

[Reference] Sridhara, Giriprasad, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. "Towards automatically generating summary comments for java methods." In Proceedings of the IEEE/ACM international conference on Automated software engineering, pp. 43-52. 2010. Link.

[Reference] Hu, Xing, Ge Li, Xin Xia, David Lo, and Zhi Jin. "Deep code comment generation with hybrid lexical and syntactical information." Empirical Software Engineering 25, no. 3 (2020): 2179-2217. Link.

[Reference] LeClair, Alexander, Sakib Haque, Lingfei Wu, and Collin McMillan. "Improved code summarization via a graph neural network." In Proceedings of the 28th international conference on program comprehension, pp. 184-195. 2020. Link.

Flexible views and layouts (with Tudor Gîrba)

[**Required**] Gîrba, Tudor. Moldable Development by Example. Talk. Link. **Instructions**: Watch the first 20 minutes; you can skip the Q&A at the end.

[**Required**] Gîrba, Tudor. Moldable Development: Guiding Technical Decisions without Reading Code. InfoQ. 2022. Link.

[Reference] Making Systems Explainable — VISSOFT 2022 Keynote

[Reference] Bragdon, Andrew, Robert Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola Jr. "Code bubbles: a working set-based interface for code understanding and maintenance." In

Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2503-2512. 2010. Link.

[Reference] DeLine, Robert, Andrew Bragdon, Kael Rowan, Jens Jacobsen, and Steven P. Reiss. "Debugger canvas: industrial experience with the code bubbles paradigm." In 2012 34th International Conference on Software Engineering (ICSE), pp. 1064-1073. IEEE, 2012. Link.

[Reference] Wang, Zijie J., Katie Dai, and W. Keith Edwards. "StickyLand: Breaking the Linear Presentation of Computational Notebooks." In CHI Conference on Human Factors in Computing Systems Extended Abstracts, pp. 1-7. 2022. Link.

[Reference] What is Max? [MaxMSP audio dataflow programming environment] Link.

Proofs and proof engineering I

[**Required**] Pit-Claudel, Clément. "Untangling mechanized proofs." In Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, pp. 155-174. 2020. Link.

[Supplemental] Jackson, Paul B. "Dynamic Proof Presentation." In Mathematical Reasoning: The History and Impact of the DReaM Group, pp. 63-86. Springer, Cham, 2021. Link.

[Supplemental] Melcer, Daniel, and Stephen Chang. "ProofViz: An Interactive Visual Proof Explorer." In International Symposium on Trends in Functional Programming, pp. 116-135. Springer, Cham, 2021. Link.

[Reference] [CoqPL'22] Coq meets literate programming: tools for documenting, preserving…

[Reference] For those not familiar with proof assistants, I encourage you to watch these videos demonstrating two widely-used modern proof assistants.

- Infinitude of primes --- a Lean theorem prover demo
- Introduction to the Coq Proof Assistant - Andrew Appel

Proofs and proof engineering II

[**Required**] Ayers, Edward. "A Tool for Producing Verified, Explainable Proofs." PhD dissertation, University of Cambridge, 2022. Link. **Instructions**: Read the abstract, and Sections 1.1, 1.3, and Chapters 5. Skim Chapter 6 and Appendix B.

[Reference] Lean Together 2021: Widgets: interactive output in VSCode

Notation comprehension aids

[**Required**] Crichton, Will. A New Medium for Communicating Research on Programming Languages. HATRA @ SPLASH. 2021. Link. **Instructions**: Read the full article, including the embedded research article on program slicing. Try your best to understand the research article. Pay close attention to the degree to which the tool helps you understand the notation, and the extent to which it does not.

[Supplemental] Head, Andrew, Amber Xie, and Marti A. Hearst. "Math Augmentation: How Authors Enhance the Readability of Formulas using Novel Visual Design Practices." In CHI Conference on Human Factors in Computing Systems, pp. 1-18. 2022. Link.

[Reference] Head, Andrew, Kyle Lo, Dongyeop Kang, Raymond Fok, Sam Skjonsberg, Daniel S. Weld, and Marti A. Hearst. "Augmenting scientific papers with just-in-time, position-sensitive definitions of terms and symbols." In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-18. 2021. Link. (Also see an online Wizard-of-Oz'd demo of the interaction here).

[Reference] Sanchez-Lengeling, Benjamin, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. "A gentle introduction to graph neural networks." Distill 6, no. 9 (2021): e33. Link.

[Reference] Wolfram, Stephen. Logic, Explainability and the Future of Understanding. (2018). Link.

[Reference] Alcock, Lara, and Nicola Wilkinson. "e-Proofs: Design of a resource to support proof comprehension in mathematics." (2011). Link.

[Reference] Hogben, Lancelot Thomas. Mathematics in the Making. Garden City, NY: Doubleday, 1960. In print.

[Reference] Cajori, Florian. A history of mathematical notations. Vol. 1. Courier Corporation, 1993. In print.

[Reference] GitHub - k-qy/notation: Collection of quotes on notation design & how it affects thought.

[Reference] Gobert, Camille, and Michel Beaudouin-Lafon. "i-LaTeX: Manipulating Transitional Representations between LaTeX Code and Generated Documents." In CHI Conference on Human Factors in Computing Systems, pp. 1-16. 2022. Link.

[Reference] Wadler, Philip. "Call-by-value is dual to call-by-name." In Proceedings of the eighth ACM SIGPLAN international conference on Functional programming, pp. 189-201. 2003.

[Reference] Ahmed, Amal. "Verified compilers for a multi-language world." In 1st Summit on Advances in Programming Languages (SNAPL 2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Holes are ~~Errors~~ TODOs (with David Moon)

[**Required**] Moon, David, Andrew Blinn, and Cyrus Omar. "tylr: a tiny tile-based structure editor." In Proceedings of the 7th ACM SIGPLAN International Workshop on Type-Driven Development, pp. 28-37. 2022. Link.

[**Required**] David Moon's Twitter thread demo'ing tylr. Link.

[Supplemental] Omar, Cyrus, Ian Voysey, Michael Hilton, Joshua Sunshine, Claire Le Goues, Jonathan Aldrich, and Matthew A. Hammer. "Toward semantic foundations for program editors." arXiv preprint arXiv:1703.08694 (2017). Link.

[Reference] Omar, Cyrus, David Moon, Andrew Blinn, Ian Voysey, Nick Collins, and Ravi Chugh. "Filling typed holes with live GUIs." In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pp. 511-525. 2021. Link.

[Reference] Omar, Cyrus, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. "Live functional programming with typed holes." Proceedings of the ACM on Programming Languages 3, no. POPL (2019): 1-32. Link.

[Reference] Zhang, Tesla. REPLs with Typed Holes. Blog post. Link.

[Reference] HaskellWiki contributors, "GHC/Typed holes," HaskellWiki. Link.

[Reference] Quick Guide to Editing, Type Checking and Compiling Agda Code. Link.

Prototyping tools for data scientists (with Philip Guo)

Note: Your questions for Philip may be the required readings, **or** about any of the other papers that we have read from Philip's group in the rest of this semester (you have already had two of his papers as required papers, and half a dozen others as supplemental readings).

[**Required**] Guo, Philip J., and Dawson Engler. "Towards practical incremental recomputation for scientists." In Workshop on the Theory and Practice of Provenance. 2010. Link. (Also see this link to the accepted paper: Link).

[Supplemental] Guo, Philip J., and Dawson Engler. "Using automatic persistent memoization to facilitate data analysis scripting." In Proceedings of the 2011 International Symposium on Software Testing and Analysis, pp. 287-297. 2011. Link.

Also see Philip Guo's papers from prior weeks.

Collaboration (with April Wang)

[**Required**] Wang, April Yi, Zihan Wu, Christopher Brooks, and Steve Oney. "Callisto: Capturing the" Why" by Connecting Conversations with Computational Narratives." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-13. 2020. Link.

[Supplemental] Wang, April Yi, Anant Mittal, Christopher Brooks, and Steve Oney. "How data scientists use computational notebooks for real-time collaboration." Proceedings of the ACM on Human-Computer Interaction 3, no. CSCW (2019): 1-30. Link.

[Reference] Epperson, Will, April Yi Wang, Robert DeLine, and Steven M. Drucker. "Strategies for Reuse and Sharing among Data Scientists in Software Teams." (2022). Link.

[Reference] Oney, Steve, Christopher Brooks, and Paul Resnick. "Creating guided code explanations with chat. codes." Proceedings of the ACM on Human-Computer Interaction 2, no. CSCW (2018): 1-20.

[Reference] Adeli, Marjan, Nicholas Nelson, Souti Chattopadhyay, Hayden Coffey, Austin Henley, and Anita Sarma. "Supporting code comprehension via annotations: Right information at the right time and place." In 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 1-10. IEEE, 2020.

Natural language as code

[**Required**] Barke, Shraddha, Michael B. James, and Nadia Polikarpova. "Grounded Copilot: How Programmers Interact with Code-Generating Models." arXiv preprint arXiv:2206.15000 (2022). Link.

[Supplemental] Little, Greg, Tessa A. Lau, Allen Cypher, James Lin, Eben M. Haber, and Eser Kandogan. "Koala: capture, share, automate, personalize business processes on the web." In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 943-946. 2007. Link.

[Supplemental] Weisz, Justin D., Michael Muller, Stephanie Houde, John Richards, Steven I. Ross, Fernando Martinez, Mayank Agarwal, and Kartik Talamadupula. "Perfection not required? Human-AI partnerships in code translation." In 26th International Conference on Intelligent User Interfaces, pp. 402-412. 2021. Link.

[Reference] Vaithilingam, Priyan, Tianyi Zhang, and Elena L. Glassman. "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language

Models." In CHI Conference on Human Factors in Computing Systems Extended Abstracts, pp. 1-7. 2022. Link.

[Reference] Miller, Robert C., Victoria H. Chou, Michael Bernstein, Greg Little, Max Van Kleek, David Karger, and M. C. Schraefel. "Inky: a sloppy command line for the web with rich visual feedback." In Proceedings of the 21st annual ACM symposium on User interface software and technology, pp. 131-140. 2008. Link.

[Reference] Little, Greg, and Robert C. Miller. "Translating keyword commands into executable code." In Proceedings of the 19th annual ACM symposium on User interface software and technology, pp. 135-144. 2006. Link.

Further reading

What follows is a small selection of readings that relate to the goal of producing beautiful, understandable programs that we did not have time to get to during this course.

Verifiable documentation

Mehrpour, Sahar, Thomas D. LaToza, and Hamed Sarvari. "RulePad: interactive authoring of checkable design rules." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 386-397. 2020. Link.

Lee, Seonah, Rongxin Wu, Shing-Chi Cheung, and Sungwon Kang. "Automatic detection and update suggestion for outdated API names in documentation." IEEE Transactions on Software Engineering 47, no. 4 (2019): 653-675. Link.

Documentation engineering

Oman, Paul W. and Curtis R. Cook. “Typographic Style is More than Cosmetic.” Communications of the ACM 33.5 (1990), pp. 506–520. Link.

Dagenais, Barthélémy, and Martin P. Robillard. "Creating and evolving developer documentation: understanding the decisions of open source contributors." In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering, pp. 127-136. 2010. Link.

Explorable explanations

Lau, Sam, and Philip J. Guo. "Data Theater: A live programming environment for prototyping data-driven explorable explanations." In Workshop on Live Programming (LIVE). 2020. Link.

Victor, Bret. "Explorable explanations." Bret Victor 10 (2011). Link.