Papercode: Generating Paper-Based User Interfaces for Code Review, Annotation, and Teaching

Priyan Vaithilingam
Harvard University
Cambridge, MA, USA

Julia M. Markel
UC San Diego
La Jolla, CA, USA

Philip J. Guo
UC San Diego
La Jolla, CA, USA

ABSTRACT

Paper can be a powerful and flexible user interface that lets programmers read through large amounts of code. Using off-the-shelf equipment, how can we generate a paper-based UI that supports code review, annotation, and teaching? To address this question, we ran formative studies and developed *Papercode*, a system that formats source code for printing on standard paper. Users can interact with that code on paper, make freehand annotations, then transfer annotations back to the computer by taking photos with a normal phone camera. Papercode optimizes source code for on-paper readability with tunable heuristics such as code-aware line wraps and page breaks, quick references to function and global definitions, moving comments and short function calls into margins, and topologically sorting functions in dependency order.

Author Keywords

paper-based interfaces, code review, teaching programming

INTRODUCTION

Paper is a flexible and powerful user interface: It is low-cost, portable, high-resolution, spreadable, stackable, foldable, shareable, and enables freehand annotations [11]. Due to these benefits, expert programmers sometimes print out their code on paper to review and annotate it. For instance, Joshua Bloch (a core developer of the Java platform) mentioned in an interview, *"the most important [debugging] tools for me are still my eyes and my brain. I print out all the code involved and read it very carefully. [...] I usually print everything out and sit on the floor surrounded by the printout, writing notes on it."* [10] Paper can be useful to programmers because, although they cannot edit code on paper, in practice they spend significant amounts of time navigating, reading, and understanding existing code [6, 8, 9]. But despite these benefits of paper, there is currently no good way to render code on paper beside printing it verbatim from an IDE or website (e.g., GitHub). Unlike textual prose, code has a complex non-linear structure with functions, classes, and global variables defined out of order and interleaved with documentation comments, so just printing it verbatim is suboptimal.

UIST '20 Adjunct, October 20–23, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7515-3/20/10.
https://doi.org/10.1145/3379350.3416191

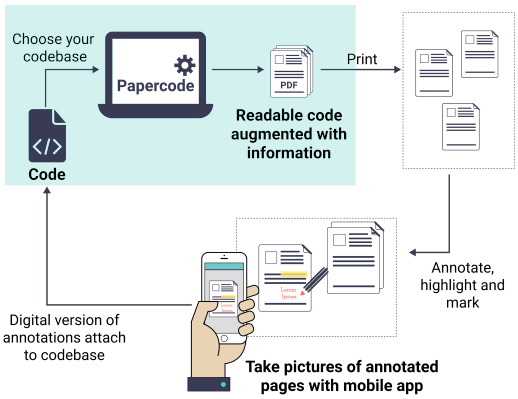

Figure 1: Papercode optimizes code for printing and allows users to scan on-paper annotations back into the computer using a phone camera.

Using off-the-shelf equipment, how can we generate a paper-based UI that supports code review, annotation, and teaching? To address this question, we developed *Papercode*, a system that formats source code for printing on standard-sized paper. Users can interact with that code on paper, make handwritten annotations, then transfer annotations back to the computer by taking photos with a normal phone camera (Figure 1). Papercode optimizes code for on-paper readability with tunable heuristics such as code-aware line wraps and page breaks, quick references to function and class definitions, topologically sorting functions in dependency order, and moving comments and short function calls into margins.

RELATED WORK

Papercode lies at the intersection of two major lines of HCI research: paper-based interfaces and code navigation tools. Researchers have long recognized the flexibility of paper as an information medium and built tools to augment it. One approach, taken by systems such as PADD [5], PapierCraft [7], CoScribe [12], and ButterflyNet [14] uses digital pens to capture handwritten annotations and gestures to synchronize with notetaking apps. Realtalk extends these ideas to programming by using projectors and cameras in a room-sized setup [13]. In contrast, Papercode does not require any special equipment; it relies only on a printer and phone camera.

Papercode was also inspired by IDE enhancements that help programmers navigate code: e.g., Code Bubbles [1], Code Canvas [3], and Debugger Canvas [2] break up codebases into functions and classes then lay them out on a 2D canvas. These systems are all computer-based interfaces; we extend some of their ideas to paper-based UIs for code navigation.

FORMATIVE STUDIES: USE CASES AND DESIGN GOALS

We conducted three small formative studies to show use cases and formulate design goals for printing code on paper.

Expert Anecdotes: The book *Coders at Work* featured interviews with 15 well-known expert programmers [10]. 7 of 15 mentioned printing code on paper for use cases such as debugging and code reviews: e.g., *"At each meeting, someone's responsible for reading their code [...] We get everybody around the table; everybody gets a stack of paper [printouts]"* (Crockford). Commonly-cited benefits of paper were 1) getting a holistic big-picture view away from the computer, and 2) being able to hand-write annotations and drawings: *"Well I did, in fact, print out the code on paper. I would sit at a desk and read it. And very often mark it up and make annotations and ask myself questions [...] and tracing it"* (Steele).

Online Forum Posts: We also found a StackExchange forum thread on code printing [4] with over 75 posts/comments. Forum users mentioned additional benefits including: *"You can put more pages side-by-side on a large conference table than on the computer screen. And you don't get distracted by Twitter or email."* But since IDEs simply print code as plain text, some mentioned annoyances of navigating through numerous pages: *"Stapling them together doesn't flow as well, and not stapling results in loose sheets that get mixed up."*

Personal Teaching Experiences: To get an educator's perspective, we asked the second author to reflect on her experiences as an undergraduate CS peer tutor. During tutoring sessions, she often explained the control and data flow of students' code by pointing to their laptop screens and verbally describing which data values flowed where. Since it was a best-practice to let students navigate their own code rather than doing it for them, and many were not adept at using IDEs, it could take a long time to jump between several layers of function/class definitions and uses, which often spanned multiple files. She also supplemented her explanations by hand-drawing sketches on notebook paper, which consist of written-out sequences of function calls along with data structure visualizations. Printing code on paper could have eased these interactions by letting students more quickly navigate their codebase and draw annotations of control and data flow directly over the code instead of in a separate notebook.

The above use cases for both professional programmers and teachers inspired a set of design goals for Papercode:

- **D1**: Papercode should work with standard commodity printers and paper, and not require any custom equipment.
- **D2**: It should support fast on-paper navigation through a realistically-sized codebase spanning a variety of files.
- **D3**: It should support freehand annotation using a regular pen and then tie those annotations back to the codebase.

PAPERCODE SYSTEM OVERVIEW

Papercode takes a codebase from a GitHub repository and parses it using a language-specific parser. Then it extracts the code for each function and formats it for printing. Figure 2 shows how it applies tunable heuristics to optimize layout:

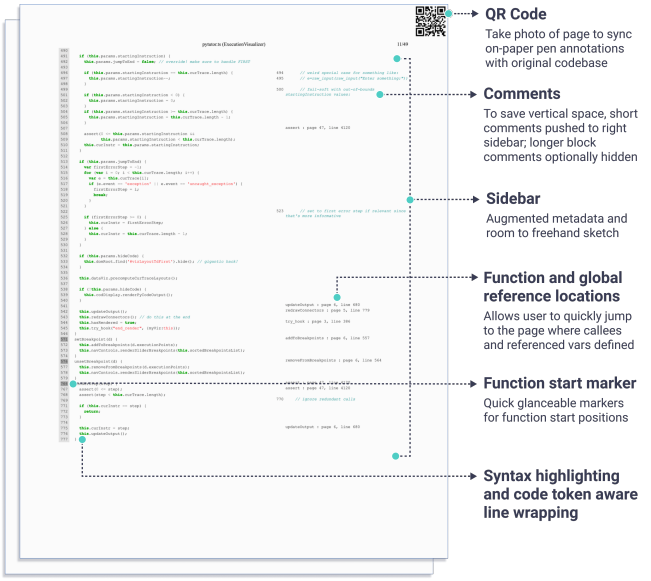

Figure 2: Example page of source code printed using Papercode.

- The right 1/3 of the page is a large sidebar reserved for metadata and freehand annotations. Long lines of code are automatically wrapped at code token boundaries.
- To save vertical space and focus attention on the code itself, it extracts out short code comments and pulls them into the right margin alongside the original code. (The user can also hide longer block comments to save even more space. Those are usually licensing metadata or auxiliary info.)
- To further save space, the user can selectively ignore certain files/functions/classes to not print them out.
- It statically detects function/method calls and adds a note in the margin showing the page number where the callee function/method is defined, to facilitate quick navigation.
- For callees that are short functions, it *inlines* the function body into the right margin so that readers can read it inline.
- It inserts page breaks at function and class boundaries so that their definitions never start in the middle of a page.
- It supports different paper and font sizes. It is up to the user to balance readability with the amount of paper they want to use. There is also syntax highlighting for color printers.
- It puts a QR code at the upper right corner of each page so that annotations can be captured with a camera (see below).
- It can also print functions in approximate topological (or reverse topological) order based on a call graph, which lets users read top-down from callers to callees (or vice versa).

Once the stack of paper for a codebase has been printed, the user can make annotations on paper with a normal pen and take photos of it with a phone camera. The QR code at the upper right links each page back to the codebase; since line numbers are at known vertical locations, Papercode can detect which lines of code the annotations were drawn near, so those annotations can appear in the IDE to reference while coding.

In sum, Papercode lets users access the flexibility of paper as a medium for reading, annotating, and teaching about code.

Acknowledgments: Thanks to Jonathan Edwards for feedback. This material is based upon work supported by the National Science Foundation under Grant No. NSF IIS-1845900.

REFERENCES

[1] Andrew Bragdon, Robert Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola. 2010. Code Bubbles: A Working Set-Based Interface for Code Understanding and Maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2503–2512. DOI: **http://dx.doi.org/10.1145/1753326.1753706**

[2] Robert DeLine, Andrew Bragdon, Kael Rowan, Jens Jacobsen, and Steven P. Reiss. 2012. Debugger Canvas: Industrial Experience with the Code Bubbles Paradigm. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*. IEEE Press, 1064–1073.

[3] Rob DeLine and Kael Rowan. 2010. Code Canvas: Zooming towards Better Development Environments. In *Proceedings of the International Conference on Software Engineering (New Ideas and Emerging Results)*. Association for Computing Machinery, Inc.

[4] StackExchange (Software Engineering). 2011. Is it common to print out code on paper? **https://softwareengineering.stackexchange.com/questions/35471/is-it-common-to-print-out-code-on-paper**. (Aug. 2011). Accessed: 2020-07-20.

[5] François Guimbretière. 2003. Paper Augmented Digital Documents. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. Association for Computing Machinery, New York, NY, USA, 51–60. DOI: **http://dx.doi.org/10.1145/964696.964702**

[6] Amy 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 Trans. Softw. Eng.* 32, 12 (Dec. 2006), 971–987. DOI: **http://dx.doi.org/10.1109/TSE.2006.116**

[7] Chunyuan Liao, François Guimbretière, Ken Hinckley, and Jim Hollan. 2008. Papiercraft: A Gesture-Based Command System for Interactive Paper. *ACM Trans. Comput.-Hum. Interact.* 14, 4, Article 18 (Jan. 2008), 27 pages. DOI: **http://dx.doi.org/10.1145/1314683.1314686**

[8] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer: An Investigation of How Developers Spend Their Time. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (ICPC '15)*. IEEE Press, 25–35.

[9] David J. Piorkowski, Scott D. Fleming, Irwin Kwan, Margaret M. Burnett, Christopher Scaffidi, Rachel K.E. Bellamy, and Joshua Jordahl. 2013. The Whats and Hows of Programmers' Foraging Diets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 3063–3072. DOI: **http://dx.doi.org/10.1145/2470654.2466418**

[10] Peter Seibel. 2009. *Coders at Work* (1st ed.). Apress, USA.

[11] Abigail J. Sellen and Richard H.R. Harper. 2003. *The Myth of the Paperless Office*. MIT Press, Cambridge, MA, USA.

[12] Jürgen Steimle, Oliver Brdiczka, and Max Muhlhauser. 2009. CoScribe: Integrating Paper and Digital Documents for Collaborative Knowledge Work. *IEEE Trans. Learn. Technol.* 2, 3 (July 2009), 174–188. DOI:**http://dx.doi.org/10.1109/TLT.2009.27**

[13] Carl Tashian. 2019. At Dynamicland, The Building Is The Computer. **https://tashian.com/articles/dynamicland/**. (Sept. 2019). Accessed: 2020-07-20.

[14] Ron Yeh, Chunyuan Liao, Scott Klemmer, François Guimbretière, Brian Lee, Boyko Kakaradov, Jeannie Stamberger, and Andreas Paepcke. 2006. ButterflyNet: A Mobile Capture and Access System for Field Biology Research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. Association for Computing Machinery, New York, NY, USA, 571–580. DOI: **http://dx.doi.org/10.1145/1124772.1124859**