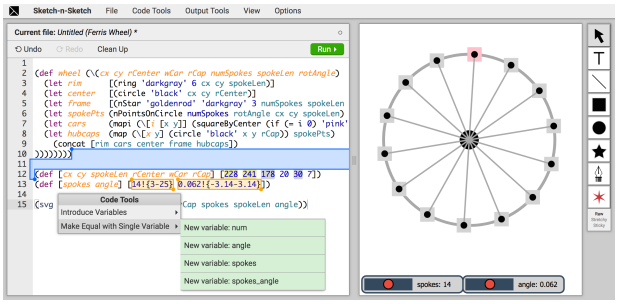
User interfaces for programming have evolved remarkably little over the decades: the programmer types source code into a text box, and an irreversible pipeline then compiles and executes the code to produce some output. This workflow (a) limits the creativity and pace of expert programmers and (b) shuts out millions more users that, though not programmers, nonetheless create a variety of complex digital objects using graphical user interface (GUI) based software. This is detrimental because computer programs—and those who write them—play an increasingly influential role in society.



`http://ravichugh.github.io/sketch-n-sketch/`

My research aims to rethink user interfaces for programming, to bridge the gap between programming languages and GUIs—a goal which sits at the boundaries of **programming languages**, **software engineering**, and **human-computer interaction**. Over the past few years, my research group and I have developed an experimental programming environment called SKETCH-N-SKETCH (shown above) for creating Web applications (HTML) and Scalable Vector Graphics (SVG). In a departure from traditional programming, SKETCH-N-SKETCH supports three novel capabilities.

- **Bidirectional Programming.** To change the output of a program in a traditional programming language, the user must edit the source code, compile and run it again, and view the new output, often repeating this loop ad nauseam. In SKETCH-N-SKETCH, the user directly manipulates the program output; our *bidirectional evaluator* then runs the program "in reverse," synthesizing small program repairs so that the new program evaluates to the desired output.[1] For situations with many direct manipulation actions, SKETCH-N-SKETCH maintains run-time traces during evaluation to facilitate faster program repair.[2] We are working to extend our techniques to handle larger classes of output changes and corresponding program repairs.
- **Output-Directed Programming.** It is often difficult to imagine what changes to the (abstract) code will, when re-evaluated, produce desired (concrete) values. In SKETCH-N-SKETCH, the user draws new elements directly in the canvas (as in traditional GUI editors), and the system automatically generates high-level, readable code that, when executed, produces that value; this starter code can be used for subsequent and more complex tasks. As the design matures, the user declares new relationships among the output values with GUI actions—e.g., to equate colors, to relate positions, or to group elements—and SKETCH-N-SKETCH semi-automatically updates the program to satisfy the declared relationships.[3] While several of these output-directed programming interactions are—by necessity—domain-specific, others can be adapted to general-purpose programming settings. To facilitate the construction of more complex programs, we are currently extending our techniques to display intermediate execution products—in addition to final output values—for users to directly manipulate.
- **Structured Text Editing.** The SKETCH-N-SKETCH code editor augments text with (i) clickable widgets directly atop the program text that allow the user to *structurally select* subexpressions and other relevant features of the program structure, and (ii) a *context-sensitive menu of program transformations* based on the current selections. This user interface streamlines the benefits of unrestricted text and structured program transformations.[4] Because programs are incomplete (i.e., they do not parse or typecheck) for long stretches during program development, we developed a semantics for running incomplete programmers, to provide live, immediate feedback throughout the development process.[5] We are currently scaling our approach to the full SKETCH-N-SKETCH system.

Together, these features allow the user to create complex, reusable programs for certain classes of digital objects, using fewer text-edits than in traditional programming languages and fewer mouse-edits than in traditional GUIs.

In the next few years, I plan to incorporate our techniques in several classroom settings. First, I will design an **introductory graphic design course** using SKETCH-N-SKETCH for University students in visual arts. These students often learn Processing to programmatically generate graphic designs. We seek to understand whether our new user interfaces can help learn how to program, compared to conventional languages like Processing. Second, I will design an **introductory programming course** using SKETCH-N-SKETCH. Nearly everyone uses GUI-based "office" applications to create papers, posters, presentations, schedules, budgets, *etc.* Millions of spreadsheet users even write basic formulas to do such work. Why not scale this up to other office applications besides spreadsheets, and to general-purpose programming integrated with GUIs? Our curriculum will explore the idea of interactively designing programs to tackle a variety of office tasks. Lastly, I will teach an **advanced functional programming course** using SKETCH-N-SKETCH to computer science majors. The basic capabilities of bidirectional programming, output-directed programming, and structured text editing aim to make certain aspects of programming more accessible to novices. In this advanced course, our most complex techniques for abstraction and program manipulation will be explored. Deploying our research advances in classroom settings will help evaluate the degree to which our ideas are successful, and will uncover additional challenges and avenues for devising new user interfaces for programming.

---

[1] Mayer, Kunčak, Chugh. **Bidirectional Evaluation with Direct Manipulation.** *Object-Oriented Programming Lang., Systems, and Applications (OOPSLA)*, 2018.
[2] Chugh, Hempel, Spradlin, Albers. **Programmatic and Direct Manipulation, Together at Last.** *Programming Language Design and Implementation (PLDI)*, 2016.
[3] Hempel, Chugh. **Semi-Automated SVG Programming via Direct Manipulation.** *User Interface Software and Technology (UIST)*, 2016.
[4] Hempel, Lubin, Lu, Chugh. **Deuce: A Lightweight User Interface for Structured Editing.** *International Conference on Software Engineering (ICSE)*, 2018.
[5] Omar, Voysey, Chugh, Hammer. **Live Functional Programming with Typed Holes.** Draft, July 2018.