IDEs for Ideas

Research Statement
Andrew Head

December 21, 2020

What do data scientists, software engineers, authors of technical tutorials, and writers of scientific papers have in common?

They all routinely read and write complex information artifacts. For instance, data scientists prototype code in computational notebooks that mix together code snippets, documentation, and inline results. Writers of scientific papers author articles that are packed with technical symbols and terms. Although these artifacts are critical for experts to share their knowledge with their audiences, they take considerable effort to compose and comprehend.

In my research, I build *IDEs for Ideas*: systems to help programmers, data scientists, and scientists read and write complex information artifacts like tutorials, notebooks, and papers (Figure 1). The research borrows from software engineering the notion of an *IDE*, or integrated development environment, a tool that supports complex writing and reading tasks by providing rich interactive features grounded in dedicated analysis algorithms.

I build and assess three types of systems. First, *program distillation tools* that help programmers, data scientists, and teachers transform messy code into readable, reusable programs (Section 1). Second, *intelligent reading tools* that help scientists understand technical terms and symbols in scientific papers (Section 2). Third, *scalable educational interactions* that connect learners to expert knowledge when they read and complete assignments (Section 3).

The broader impacts of my work reach both academia and industry. **Four of my keystone papers on this topic received paper awards at premier ACM and IEEE conferences in HCI and programming tools**. I have received support from an Alfred P. Sloan Foundation Grant, an NDSEG Fellowship, and a grant from the Allen Institute for AI. The research has been grounded in real-world contexts with research internships on programming tools research teams at Google and Microsoft Research. **One tool has been adopted by Microsoft as a feature into their widely-used VSCode code editor**.

As a systems researcher in human-computer interaction, my methods involve the human-centered design and implementation of novel interactions in tandem with algorithms that enable them. I follow an artifact-driven approach to research described by Wobbrock et al.,[1] wherein contributions of the research come from formative user

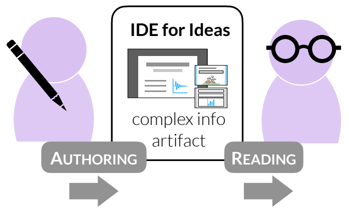

Figure 1: An *IDE for Ideas* is an intelligent interactive tool that aids in the authoring or reading of a complex information artifact like a programming tutorial, computational notebook, or scientific paper.

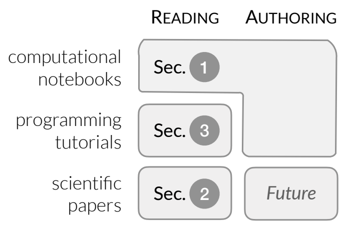

Figure 2: A visual guide to the topics in this research statement.

[1] Wobbrock and Kientz. "Research Contributions in Human-Computer Interaction". In: *ACM Interactions* 23.3 (2016)

research (i.e., interviews, document analyses, observations), usable tools that embody new design ideas, and evaluations of those tools with users. The systems incorporate algorithms for program slicing, program synthesis, and natural language definition detection. The algorithms are tailored to support user needs, and in turn the interactions motivate improvements to the algorithms.

1 Interactive Systems for Distilling Programs in Software Development and Data Science

In software engineering, data science, and the programming classroom, a good sample program is a powerful resource for conveying knowledge. My dissertation explored how *program distillation tools*, a type of IDE for Ideas, could help programmers create readable, reusable sample programs by providing novel authoring capabilities supported by static and dynamic program analysis.

My research focused on three case studies: data scientists cleaning their computational notebooks; programmers sharing snippets; and teachers creating step-by-step, output-rich programming tutorials. My formative observations and interviews with users from all of these groups revealed that programmers could be supported in producing sample programs if their tools could help them *select* code of interest from sample programs, *simplify* it to remove superfluous detail, *supplement* it with descriptions and outputs, and *sequence* programs into series of interrelated snippets.[2]

For example, in the domain of data science, distillation tools help data scientists collect, review, and share fragmented analysis code. The editors data scientists use to write code—like computational notebooks and REPLs—often fragment code in a way such that it becomes difficult if not impossible to identify the code that produced promising results. I developed Code Gathering Tools as a distillation tool that **augments computational notebooks to help data scientists collect, review, and share fragmented analysis code.**[3]

The key insight is that data scientists could be supported by applying a tried-and-true program analysis method—program slicing—to a notebook's execution log aligned with the notebook's cells (Figure 3). I implemented a static program slicer for Python. Then I built a user interface that supported selection of results in the notebook and (1) extracting ordered, cleaned notebooks that produced those results (Figure 4); (2) finding code that produced a result; and (3) comparing slices of code that produced versions of results.

Code gathering tools have seen impact in academia and industry. Our paper on the topic received a **best paper award at CHI, ACM's**

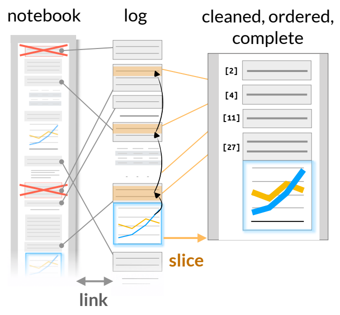

Figure 3: To help data scientists clean and recover analysis code in computational notebooks, Code Gathering Tools slice an execution log linked to the notebook.

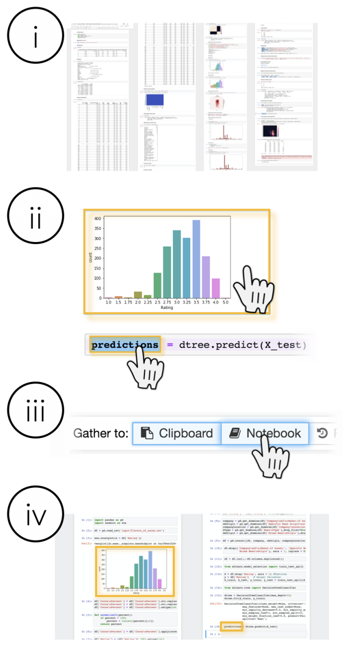
i
ii
predictions = dtree.predict(X_test)
iii
Gather to: Clipboard Notebook
iv

Figure 4: Cleaning analysis code in a notebook with Code Gathering Tools.

[2] Head. "Interactive Program Distillation". PhD thesis. UC Berkeley, 2020

[3] Head, Hohman, Barik, Drucker, and DeLine. "Managing Messes in Computational Notebooks". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, 2019. (Demo video). **Best Paper Award**.

premier conference in HCI. The project sparked follow-on research, including a Master's thesis project,[4] a Microsoft summer internship project, and a UC Berkeley undergraduate independent study project. In industry, a qualitative usability study showed that professional data scientists found the system useful and appropriating the tools for new purposes during exploratory data analysis. **Microsoft incorporated the tools into their Python extension in Visual Studio Code, where it has been installed over four thousand times.**

To extend the design space, I designed **two interactive systems to help programmers author tutorials through iterative interaction with algorithms for program analysis**. CodeScoop helps programmers extract brief, complete snippets from existing programs through incremental selection of code with a program slicer (Figure 5) and simplification of code by substituting program expressions with concrete literal values collected during program execution (Figure 6).[5] Torii helps teachers create step-by-step programming tutorials containing *many* interrelated snippets and outputs with a novel computational notebook environment with live programming that propagates edits across snippets, source programs from which they are made, and outputs generated from running the snippets (Figure 7).[6]

In lab studies, I've found that the tools support tutorial authoring: with CodeScoop, for instance, programmers extract snippets in half the time it takes with a baseline editor. The design of the tools were grounded in interviews with expert authors, observations, and a content analysis of hundreds of tutorials. My dissertation[2] (Chapters 2, 3) contributes a literature review on program comprehension, sample program design, interactive tools, and program analyses to set an agenda for future research into IDEs for Ideas that support program distillation for tutorials and data science.

2 Intelligent Interfaces for Reading Scientific Papers

In my current work, **I explore how IDEs for Ideas can support reading of complex natural language artifacts**. Intelligent reading interfaces pose a unique challenge for researchers in computer science. To minimize disruption, interface elements need to be informative, yet small in size and easy to dismiss. Interactions need to be developed concurrently with algorithms such that algorithms provide useful output and the interactions provide suitable error recovery mechanisms.

I and Professor Marti A. Hearst began study of this problem by launching the ScholarPhi project,[7] securing a grant from the Alfred P. Sloan Foundation. I designed and assessed ScholarPhi, an interactive, intelligent system for reading scientific papers. Motivated by my formative observations of readers, ScholarPhi was designed to help scientists

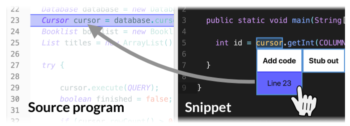

Figure 5: Interactive, iterative code selection with CodeScoop.

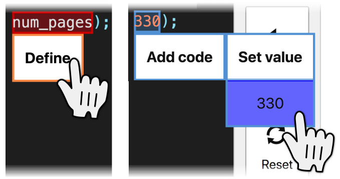

Figure 6: Simplifying a code snippet with CodeScoop by replacing an expression with a value from the execution trace.

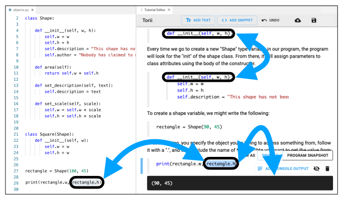

Figure 7: Authoring a programming tutorial in Torii, a live programming editor that supports linked editing of snippets, source programs, and outputs.

[4] Chaudhary. "Jupyter's Archive: Searchable Output Histories for Computational Notebooks". MA thesis. 2019

[5] Head, Glassman, Hartmann, and Hearst. "Interactive Extraction of Examples from Existing Code". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, 2018. (Demo video). **Nominated for Best Paper Award**.

[6] Head, Jiang, Smith, Hearst, and Hartmann. "Composing Flexibly-Organized Step-by-Step Tutorials from Linked Source Code, Snippets, and Outputs". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, 2020. (Demo video). **Nominated for Best Paper Award**.

[7] Head, Lo, Kang, Fok, Skjonsberg, Weld, and Hearst. "Augmenting Scientific Papers with Just-in-Time, Position-Sensitive Definitions of Terms and Symbols". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 2021. arXiv: 2009.14237 [cs.HC]. (Demo video). To appear.

understand technical terms and symbols that they would otherwise find too costly to look up. **I built ScholarPhi with four innovative features that inform scientists with minimal disruption**: position-sensitive tooltips that expose definitions of terms and symbols; precise subsymbol selection; visual highlighting and lowlighting of passages based on whether they include selected terms; and equation diagrams that show definitions of multiple terms at once (Figure 8).

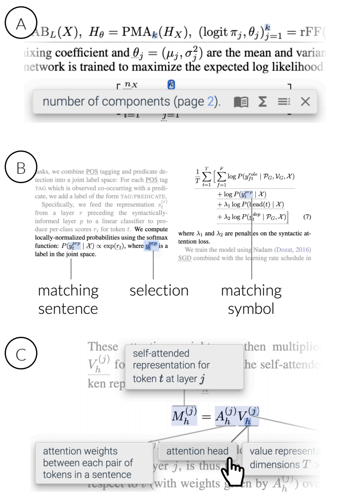
A
number of components (page 2).
B
matching sentence
selection
matching symbol
C
self-attended representation for token t at layer j
attention weights between each pair of tokens in a sentence
attention head
value representa dimensions

Figure 8: Reading a scientific paper with ScholarPhi. Features include (A) position-sensitive tooltips showing definitions of terms and symbols; (B) highlighting passages containing a term and lowlighting all others; (C) equation diagrams showing definitions of multiple symbols.

Empirical studies of the tool suggest that ScholarPhi provides useful information in a usable format without distraction. In a controlled study, researchers answered questions about a scientific paper more quickly, while viewing less of the paper, with the tools.

I and Hearst developed ScholarPhi into cross-institution, interdisciplinary research project at the crossroads of AI and HCI. Postdoctoral scholar Dongyeop Kang has begun development novel NLP algorithms to extract definitions of terms from academic texts.[8] My initial error recovery mechanisms in ScholarPhi provided a starting point for research on the topic by Ph.D. student Raymond Fok and Professor Daniel S. Weld. We actively collaborate on research with the Semantic Scholar research team at the Allen Institute for AI. An alpha release of the tool is planned. The release is backed by funding, infrastructure, and engineering from the Allen Institute, and offers a real-world setting to evaluate the algorithms and interactions.

3 Connecting Learners to Expert Knowledge at Scale

When learners face obstacles understanding complex information, they can be supported through timely interaction with experts. A third topic in my research shows how IDEs for Ideas can connect learners with expert explanations and assistance at scale.

First, tools can **augment complex artifacts with generated explanations encoding expert knowledge**. In a collaborative study with the Google Engineering Productivity Research team, I conducted firehouse interviews and experience sampling that showed that a shortage of engineering effort available can inhibit satisfactory documentation of APIs.[9] To help programmers understand unexplained code in programming documentation, I developed Tutorons, or rule-based algorithms that generated context-relevant natural language descriptions of code in tutorials. The explanations reduce programmers' need to search for supplemental documentation,[10] and demonstrate the possibility of encoding expert knowledge in natural language explanation algorithms that augment complex artifacts.

Second, they can **support dialogue between learners and experts at scale**. I helped design FixPropagator, a tool for massive program-

[8] Kang, Head, Sidhu, Lo, Weld, and Hearst. "Document-Level Definition Detection in Scholarly Documents: Existing Models, Error Analyses, and Future Directions". In: *Proceedings of the EMNLP First Workshop on Scholarly Document Processing*. 2020

[9] Head, Sadowski, Murphy-Hill, and Knight. "When Not to Comment: Questions and Tradeoffs with API Documentation for C++ Projects". In: *Proceedings of the International Conference on Software Engineering*. 2018

[10] Head, Appachu, Hearst, and Hartmann. "Tutorons: Generating Context-Relevant, On-Demand Explanations and Demonstrations of Online Code". In: *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2015. **Nominated for Best Paper Award**.

ming classrooms that lets teachers provide personalized feedback on programming assignments to many students at once. The tool uses program synthesis to learn bug-fixing program transformations to student submissions, and then leverages the transformations as vehicles to propagate teacher-written feedback to students.[11] I have also explored how tools can motivate experts to engage with learners through collaborative interactions with gameplay elements.[12]

[11] Head*, Glassman*, Soares*, Suzuki, Figueredo, D'Antoni, and Hartmann. "Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis". In: *Proceedings of the Conference on Learning at Scale*. ACM, 2017. (*) The first three authors contributed equally to this work.

[12] Head, Xu, and Wang. "ToneWars: Connecting Language Learners and Native Speakers through Collaborative Mobile Games". In: *International Conference on Intelligent Tutoring Systems*. Springer, 2014. (Demo video).

Research Agenda

In future research, I am interested in exploring how intelligent tools can address pressing knowledge-sharing problems of our time with research integrating HCI, AI, education, and programming. My goals, from most concrete to most speculative, are as follows:

Characterizing good pedagogy in programming tutorials: Programming tutorials are widely-used to communicate programming knowledge in programming classrooms, online education, and industry. The tutorial authors I speak with ask about how to design them well: How large should snippets be? Should they be interactive? What types of interactivity are effective (e.g., visualizing results, live programming, etc.)? How should tricky programming concepts be explained? I would like to conduct controlled experiments and observations to generate this knowledge, disseminate findings as guidelines to tutorial authors in the academy and industry, and develop new tutorial authoring toolkits that encode best practices.

Data analysis tools that are safe, fast, and presentation-friendly: Recently, research and industry have proposed an abundance of data analysis tools with novel execution models like Jupyter, Observable, Streamlit, and my collaborators' NBSAFETY.[13] These tools implicitly make tradeoffs between rapid exploration, safety (i.e., preventing loss of results), and presentation. I would like to characterize these tradeoffs through studies with data scientists, and explore new models of analysis tools that simultaneously support safety, speed, and presentation.

[13] Macke, Gong, Lee, Head, Xin, and Parameswaran. "Static Analysis for Safer Notebook Interactions". In: *Proceedings of the SPLASH Workshop on HCI and PL*. ACM, 2020

Program synthesis for programming education: Could state-of-the-art tools for generating code (e.g., program synthesis, evolutionary algorithms, and program repair techniques) be brought to bear to produce readable code for educational purposes? I would like to explore how recent techniques for generating programs for general-purpose programming languages can be extended to (1) tailor sample programs to individual readers by taking their prior knowledge into account, and (2) generate not only programs but sequences of snippets that show in the most effective way how the program can be built up line-by-line.

Intelligent writing tools for scientists: How can tools help scientists write better articles? I am interested in the challenge problem of designing intelligent writing tools to motivate the co-design of novel NLP algorithms and writing interfaces. This requires tailoring of NLP models to support definition detection, coherence measurement, coreference resolution, and text generation, among other things. It also requires understanding interaction patterns for humans working together with text generators of specialized text.

Beyond the goals above, I am broadly interested in exploring how algorithms and interactions can help people create and read complex information artifacts of all kinds, including infographics, interactive data narratives, textbooks, mathematical proofs, and others. I look forward to research collaborations that can characterize the process of reading and writing in these domains and contribute new algorithms and interactions to support those processes.