ARA Proposal: TYCHE: An IDE for Property-Based Testing

PI: Benjamin C. Pierce, Professor, Computer and Information Science, University of Pennsylvania
Co-PI: Andrew Head, Assistant Professor, Computer and Information Science, University of Pennsylvania

Cash funding needed: $80,000
AWS Promotional Credits needed: $5,000

Amazon contacts:
Michael Hicks, AWS Senior Principal Scientist, mwhicks@amazon.com
Sean McLaughlin, AWS Principal Scientist

Abstract

Property-based testing (PBT) is an approach to rigorous software testing that makes high-level, specification-based reasoning available to everyday developers. Although PBT is increasingly making its way into software development workflows at Amazon and other major software firms, current tools are limited by their reliance on outdated interfaces that provide little user feedback or facility for interactive experimentation.

We propose TYCHE, an integrated development environment (IDE) for PBT that gives developers in-editor feedback and assistance for PBT.

Our current prototype of TYCHE integrates with Hypothesis, the most popular PBT library in Python, and it focuses on helping developers answer a single question: How well did a particular test exercise the program under test? Using this Research Award, we plan to evolve this prototype into a powerful, generic tool with real-world users, including at the Cedar group and others within AWS. Specifically, we will (1) generalize TYCHE beyond Python, providing a language-agnostic interface and carving a path for integration with PBT libraries in other languages, beginning with Rust. At the same time, we will (2) make TYCHE more powerful, adding tools for reasoning about failures, stress-testing existing test suites, and potentially interactively synthesizing test harnesses. Finally, we will (3) evaluate these tools with real users. The ultimate aim is to put the power of PBT at developers' fingertips, at Amazon and beyond.

Keywords: Property-based tesing, formal methods, high-assurance software, human-computer interaction

Introduction

Property-based testing is a powerful methodology and accompanying tooling that allow developers to rapidly gain confidence in formal specifications of their programs via testing—it has been called "Formal specification without formal verification." To use PBT, developers start with an executable property that encodes some desired behavior of their program. Then the system randomly generates hundreds or thousands of inputs and checks that the program's behavior satisfies the properly for all of them.

PBT has proven effective at identifying subtle bugs in a wide variety of settings, including telecommunications software [2], replicated file and key-value stores [16, 5], automotive software [3], and other complex systems [15]. A recent study from Amazon [6] described using PBT, along with related approaches to "lightweight" formal methods, to find and fix bugs in the ShardStore codebase. At the same time, PBT also synergizes beautifully with "heavier" formal methods. Teams who eventually plan to prove a program correct—for example, with the help of an SMT solver or via interactive theorem proving—can get significant value from PBT for catching bugs early, before significant effort has gone into verifying broken specifications [21, 4]. Additionally, PBT can be used to to validate high-performance real-world implementations against slower verified specifications; for example, we understand that the AWS team behind Cedar [22] leverages PBT in this capacity.

Despite these successes, currently available tools for PBT are missing a key opportunity: interactivity. With one very recent exception [10], the user interfaces of PBT provide only a command-line interface for running tests. Even unit testing frameworks, which have significantly less information-per-test to communicate, have richer interfaces. Working under these constraints means that the tools (1) cannot display the

wealth of data produced during the PBT process and (2) cannot benefit from natural user interactions. Recent studies of PBT in industry [13, 12] suggest that adding these affordances would make PBT significantly more effective and usable.

We propose TYCHE, an integrated development environment (IDE) for PBT built on VS Code. An early prototype of the interface is shown in Figure 1 and was presented as a demo at the ACM Symposium on User Interface Software and Technology (UIST) in 2023. In its current form, TYCHE is a simple tool for visualizing testing feedback. As the underlying PBT library (in this case Hypothesis [18] in Python) generates random inputs to exercise the property under test, the TYCHE interface collects statistics on these inputs and visualizes them for the user in the form of various charts. This kind of visualization is extremely useful, since a property's ability to find bugs in a given program is only as good as the inputs that are generated for it. If the generator produces too many duplicate inputs, small inputs, or inputs that do not satisfy a particular check, the property may miss bugs. TYCHE helps avoid such false negatives by surfacing easily available information about generated inputs and making it easy to interpret.

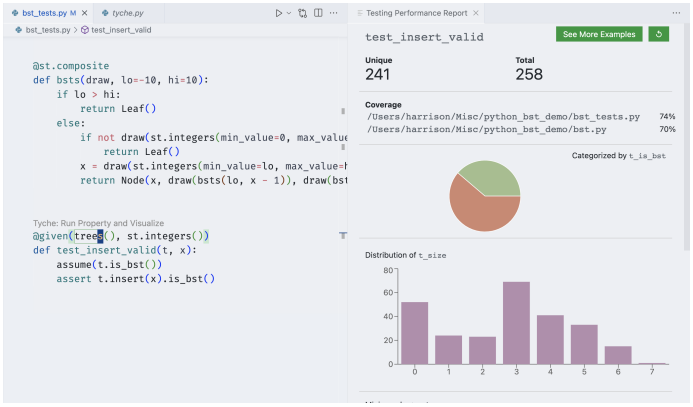

Figure 1: An early version of the TYCHE interface. Left: A Hypothesis property called `test_insert_valid`. Right: An analysis of the data used to check that property.

The current TYCHE prototype is limited in several dimensions. It only works with Hypothesis in Python, and it only helps users with one step of the PBT process—evaluating generators—with no assistance for tasks like writing properties or debugging failures. We plan to expand and generalize TYCHE to a comprehensive interactive toolkit that make the PBT process significantly more usable and useful across a variety of languages and PBT frameworks. Concretely:

1. We will generalize TYCHE to be language and framework agnostic, significantly increasing its reach. To demonstrate success, we will maintain the current integration with Hypothesis (in Python) and add one for Bolero (in Rust) [7] which is developed and used by engineers at Amazon.
2. We will expand the set of TYCHE features to include include in-depth code-coverage feedback, debugging assistance, mutation testing support, and interactive property and generator synthesis. As we go, we will continue to interact with real users to test our design.
3. We will evaluate the success of TYCHE via two small user studies, enlisting both Bolero (Rust) and Hypothesis (Python) users to try the tool and examining how it improves on existing workflows.

Academic results of this work will be published at a top-tier research venue such as UIST; the software we develop will be released and maintained as an open-source VSCode extension.

Methods

These plans will be carried out in three overlapping phases.

Phase 1: Generalizing TYCHE*'s Interface.* The first step will be to generalize TYCHE's implementation to work with a much wider array of PBT libraries, frameworks, and language ecosystems.

The current implementation of the TYCHE VSCode extension is coupled to Python: it calls out to a Python process, reads the output, and uses that output to construct a visualization. We plan to re-architect

the code do break this coupling. First, we will to add a daemon process to mediate the interactions between the VSCode extension and the TYCHE Python stub. (This daemon will be written in Rust to maximize performance and maintain memory safety.) Next, the extension and Python stub will both be modified to communicate with the daemon rather than each other directly. Finally, new stubs will be written in Rust and other languages, adapting the existing testing frameworks in those languages to communicate with the TYCHE daemon.

This process is an opportunity for some generalization and re-thinking of the data format that TYCHE uses to communicate as well. Concretely, it may be useful for intermediate data about previously run tests to be stored in a relational database (e.g., managed by SQLite [14]). This would allow the TYCHE extension to create visualizations based on optimized queries, minimizing the amount of data communicated and thus improving performance. It would also open up the opportunity for asynchronous communication; a long-running test could log data to the database, and the TYCHE interface could query and visualize that data much later.

While these steps are primarily engineering effort, they are necessary steps towards making TYCHE useful for developers like the ones on the Cedar team at Amazon, who primarily work in Rust. These steps will also enable further research and development, which we describe in the rest of this section.

Phase 2: Expanding TYCHE*'s Feature Set.* TYCHE's ultimate feature set will evolve through interaction with users, but at least the following seem critical.

Support for Debugging. A testing tool not only needs to help *identify* bugs; it should also help developers understand and repair them. Existing PBT frameworks support these processes primarily via *test-case minimization* [9, 1, 17]. When a counterexample is found for a given property, the framework attempts to "shrink" that counterexample down to the smallest input that still causes the property to fail. Shrinking is immensely useful, but more can be done. For example, shrinking a record `{x: 10, y: 0}` to `{x: 1, y: 0}` gives some information about the value of `x` that leads to an error (suggesting that it must be positive because otherwise this component would have continued shrinking to `0`), but it does not give much information about `y`. It might be that the bug is only triggered when `y = 0` or it might be that any value of `y` leads to failure.

We propose to address this problem by interactively combining test-case minimization with test-case *generalization*. After being presented a shrunk counterexample, the user can ask the system to randomly vary sub-components of that counterexample to help determine which are important for triggering the bug. In the example above, the user would click on `y: 0` and the system would re-test the property with inputs like `{x: 0, y: 100}` and `{x: 0, y: -10}` to try to determine which specific values of `y` are problematic. As they explore the counterexample in this way, they will be able to save inputs that seem important, interactively building a representative set of inputs that can be used (1) to find the bug, (2) to document the bug, and (3) as regression tests to ensure the bug does not return. Note that this process is necessarily interactive—supporting the user in their own reasoning and exploration—and thus requires an interactive tool like TYCHE.

Support for Mutation Testing. Once a test suite has stopped detecting bugs, it is good practice to apply *mutation testing* to understand how well the code is being tested. Mutation testing is a method of "testing the tests" where the user intentionally injects bugs into the program and checks that their test suite does, in fact, flag them. While automated techniques mutation testing do exist [19], these techniques have a high rate of *false positives*: they inject mutations that are not actually bugs and then complain that testing does not reject them. In contrast, tools like `pytest-mutagen` [20] give the user the power to define vustom mutations via an ergonomic interface.

We will build on ideas from tools like mutagen to add a fully featured mutation testing workflow into TYCHE. First, the user will describe mutations by simply modifying their code; as each mutant is completed, TYCHE will register it, save a diff on the side, and revert the code to the correct version. When a suite of

mutants have been assembled, TYCHE will be able to replay each diff against the test suite, tracking how long it takes to find each mutant. At the end, TYCHE will display charts to show which mutants were caught, and how easily, by leveraging *task bucket charts* used in our prior work on the ETNA evaluation platform [23]. As the codebase evolves, TYCHE will try to keep the mutants up to date, asking the user to re-implement a given mutant only when the code has changed too much.

Support for Property and Generator Authoring (Stretch Goal). The tools proposed thus far mostly concern the outputs of the PBT process, but the *inputs*—the properties under test and the random data generators that produce test inputs—are also critical to the process. We believe that the interactive nature of TYCHE could also be leveraged to help users decide which properties to test and which generators to test with. Prior work [11] has shown that language models can, indeed, be used to help define correctness conditions for functions, and we have carried out some informal experiments suggesting that models like Anthropic's LLM Claude can produce PBT generators for simple data types. We will take the next step, experimenting with adding foundation models to the TYCHE interface and building workflows that allow those models help the user build their test harness.

Of course, users can already at least experiment with using foundation models for test harness authoring in their editor with tools like GitHub Copilot—why build this functionality into TYCHE? The answer is both *integration* and *validation*. TYCHE can significantly constrain the potential outputs of the model, based on built-in knowledge about properly formatted and framed properties and generators. It can also test these properties and generators before suggesting them, discarding ones that are trivial or ill-formed. Furthermore, if the user has established some mutants, TYCHE can use these as feedback to determine which properties or generators are actually useful. We argue that the user's interactions with TYCHE can be used to refine the outputs of foundation models to the most useful results.

Phase 3: Evaluating TYCHE*'s Impact.* TYCHE is an interdisciplinary project, connecting theory and algorithms from programming languages, formal methods, and software engineering with practical interface design research from the field of human computer interaction (HCI). This approach has significant benefits [8], chief among which are good methodologies for evaluating impact from the HCI literature.

We will evaluate TYCHE via two small usability studies. First, we will recruit 20 Python developers familiar with Hypothesis and ask them to perform two short minute testing tasks. For one task, they will have TYCHE to help them, and for the other they will need to do the task manually. We will measure the time required to solve the task, and afterward we will interview the users and ask them about their experience using TYCHE. We expect the results from this study to serve as an straightforward argument that TYCHE adds value in simple scenarios. Second, we will interview 5–10 organic users of TYCHE about their experiences. (Ideally, these would be Rust/Bolero users inside Amazon, but this will require some care around intellectual property; if necessary, open-source Rust developers can be used instead.) These interviews will provide a richer picture of the way real developers use TYCHE and feed back into the design of the system.

These studies will be analyzed in an academic paper to paint a clear picture of the benefits (and perhaps drawbacks) of the TYCHE interface. Both studies will be evaluated by the University of Pennsylvania's Institutional Review Board (IRB), ensuring that the studies are ethical and of minimal legal risk.

Expected Results

The project will run for the full calendar year of 2024.

- **Q1** We will build the infrastructure necessary to generalize TYCHE to work with PBT frameworks outside of Python. The VSCode extension will be officially released as beta software, and corresponding libraries will be released in the Python and Rust ecosystems.
- **Q2** With the help and guidance of industry contacts, we will expand TYCHE's feature set to assist users throughout the PBT workflow.

Q3 We will carry out a user study to evaluate TYCHE's effectiveness, and we will begin to engage open-source collaborators to help manage the codebase and address emerging issues; these might be industrial users of TYCHE or maintainers of adjacent PBT projects. The study plan will be approved by IRB. The extension and libraries will be released as "1.0" software.

Q4 We will run the user study, collect feedback, and summarize the results in a research paper submission. We will continue working with open-source collaborators to develop on a long-term maintenance plan.

Funds Requested

We request $80,000 of cash funding and $5,000 of AWS Promotional Credits for a total of $85,000.

Cash funding. This project will partially fund tuition and stipend for one PhD student, TYCHE's lead developer Harrison Goldstein, for one year ($50,000), as well as summer funding for the PI ($15,000) and co-PI ($10,000). Additionally, we budget $5,000 for travel and conference registration expenses.

AWS Promotional Credits. We request $5,000 of funding in the form of AWS Promotional Credits to support hosting and upkeep for the project itself, as well as to support our experiments with foundation models. We plan to use $1,000 of credits for *Amazon EC2*, hosting centralized data that TYCHE needs to share between instances, including live user feedback. Additionally, we will use up to $4,000 experimenting with *Amazon Bedrock* and its foundation models, to explore how effective these models are at helping users synthesize properties and generators.

Additional information

We hope most of the following has come through clearly in the above document, but we repeat our answers to the CFP's core questions here, to aid reviewers:

1. Does your work target analysis of protocols, code, or configuration? Please provide information about your domain and the type of analysis. *Response: We target specification-based analysis of code via lightweight formal methods, specifically property-based testing.*
2. What are the current applications of your work? (e.g., libraries, codebases, or industry code). *Response: The* TYCHE *interface can be used as part of a larger testing and formal methods workflow to build trust in any kind of code, including both libraries and industrial applications. PBT has been used in industrial applications from telecommunications and automotive firmware to distributed databases, filesystems, and financial analysis.*
3. What are potential applications of your work to Amazon? *Response: PBT is also used in various places at AWS [6]. One especially natural group of users for* TYCHE*'s Rust workflow is the Cedar team, which currently relies on PBT to validate a model implementation (suitable for formal reasoning) to the real codebase. Evaluating the extent and quality of testing is crucial in this situation because insufficient testing could miss important deviations and significantly reduce the value of verifying the model.*
4. What assumptions are made by your work? If the techniques proposed are sound: What are issues that may invalidate this result? *Response: We make the same assumptions as PBT generally—namely, that users are able to write executable specifications for their code' our study of PBT in industry [12] confirms that this is often the case. One of the goals of the present project is to empower users in this.*
5. If your work involves the development and maintenance of a tool: (a) Under what license is or will your tool be released? *Response: MIT (a permissive open-source license).* (b) What on-boarding/tutorial material is available? *Response: Building a full tutorial will be part of the project activities.* (c) Is your tool actively maintained (i.e., commits within last 3 months)? *Response: Yes.* (d) How many active contributors does your project have? *Response: Currently 1; the project's main goal is to bring in both users and contributors from the Hypothesis and and Rust ecosystems to seed an active open-source community around* TYCHE.

References

[1] Thomas Arts. On shrinking randomly generated load tests. In *Proceedings of the Thirteenth ACM SIGPLAN workshop on Erlang*, Erlang '14, pages 25–31, New York, NY, USA, September 2014. Association for Computing Machinery.

[2] Thomas Arts, John Hughes, Joakim Johansson, and Ulf Wiger. Testing telecoms software with quviq QuickCheck. In *Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, ERLANG '06, pages 2–10, New York, NY, USA, September 2006. Association for Computing Machinery.

[3] Thomas Arts, John Hughes, Ulf Norell, and Hans Svensson. Testing AUTOSAR software with QuickCheck. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–4, April 2015.

[4] S. Berghofer and T. Nipkow. Random testing in Isabelle/HOL. In *Proceedings of the Second International Conference on Software Engineering and Formal Methods, 2004. SEFM 2004.*, pages 230–239, September 2004.

[5] Ann Blandford, Dominic Furniss, and Stephann Makri. Analysing Data. In Ann Blandford, Dominic Furniss, and Stephann Makri, editors, *Qualitative HCI Research: Going Behind the Scenes*, Synthesis Lectures on Human-Centered Informatics, pages 51–60. Springer International Publishing, Cham, 2016.

[6] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, Jacob Van Geffen, and Andrew Warfield. Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, pages 836–850, New York, NY, USA, October 2021. Association for Computing Machinery.

[7] Cameron Bytheway. Bolero Book, 2023.

[8] Sarah E. Chasins, Elena L. Glassman, and Joshua Sunshine. PL and HCI: better together. *Communications of the ACM*, 64(8):98–106, August 2021.

[9] Koen Claessen. Shrinking and showing functions: (functional pearl). In *Proceedings of the 2012 Haskell Symposium*, Haskell '12, pages 73–80, New York, NY, USA, September 2012. Association for Computing Machinery.

[10] Matthew C. Davis, Sangheon Choi, Sam Estep, Brad A. Myers, and Sunshine. NaNoFuzz: A Usable Tool for Automatic Test Generation. 2023.

[11] Elizabeth Dinella, Gabriel Ryan, Todd Mytkowicz, and Shuvendu K. Lahiri. TOGA: a neural method for test oracle generation. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, pages 2130–2141, New York, NY, USA, July 2022. Association for Computing Machinery.

[12] Harrison Goldstein, Joseph W Cutler, Daniel Dickstein, Benjamin C Pierce, and Andrew Head. Property-Based Testing in Practice. 2024.

[13] Harrison Goldstein, Joseph W Cutler, Adam Stein, Benjamin C Pierce, and Andrew Head. Some Problems with Properties. volume 1, December 2022.

[14] Richard D Hipp. SQLite, 2020.

[15] John Hughes. Experiences with QuickCheck: Testing the Hard Stuff and Staying Sane. In Sam Lindley, Conor McBride, Phil Trinder, and Don Sannella, editors, *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 169–186. Springer International Publishing, Cham, 2016.

[16] John Hughes, Benjamin C. Pierce, Thomas Arts, and Ulf Norell. Mysteries of DropBox: Property-Based Testing of a Distributed Synchronization Service. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 135–145, April 2016.

[17] David R. MacIver and Alastair F. Donaldson. Test-Case Reduction via Test-Case Generation: Insights from the Hypothesis Reducer (Tool Insights Paper). In Robert Hirschfeld and Tobias Pape, editors, *34th European Conference on Object-Oriented Programming (ECOOP 2020)*, volume 166 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:27, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 1868-8969.

[18] David R MacIver, Zac Hatfield-Dodds, and others. Hypothesis: A new approach to property-based testing. *Journal of Open Source Software*, 4(43):1891, 2019.

[19] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman. Mutation Testing Advances: An Analysis and Survey. *Advances in Computers*, January 2018.

[20] Timothee Paquatte and Harrison Goldstein. pytest-mutagen, 2023.

[21] Zoe Paraskevopoulou, Cătălin Hriţcu, Maxime Dénès, Leonidas Lampropoulos, and Benjamin C. Pierce. Foundational Property-Based Testing. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving*, Lecture Notes in Computer Science, pages 325–343, Cham, 2015. Springer International Publishing.

[22] Amazon Web Services. Cedar Language, 2023.

[23] Jessica Shi, Alperen Keles, Harrison Goldstein, Benjamin C Pierce, and Leonidas Lampropoulos. Etna: An Evaluation Platform for Property-Based Testing (Experience Report). *Proc. ACM Program. Lang.*, 7, 2023.

Benjamin C. Pierce, Curriculum Vitae

Capsule Bio.
Benjamin Pierce is Henry Salvatori Professor of Computer and Information Science at the University of Pennsylvania and a Fellow of the ACM. His research interests include programming languages, type systems, language-based security, computer-assisted formal verification, differential privacy, and synchronization technologies. He is the author of the widely used graduate textbooks Types and Programming Languages and Software Foundations. He has served as co-Editor in Chief of the Journal of Functional Programming, as Managing Editor for Logical Methods in Computer Science, and as editorial board member of Mathematical Structures in Computer Science, Formal Aspects of Computing, and ACM Transactions on Programming Languages and Systems, as vice-chair of ACM SIGPLAN, as a member of ACM Council, and as chair of the SIGPLAN ad hoc committee on climate change. He holds a doctorate *honoris causa* from Chalmers University and in 2021 was awarded the inaugural SIGPLAN Distinguished Educator's Award. He is also the lead designer of the Unison file synchronizer and co-developer of the Midspace virtual conference platform.

Education.
Ph.D. (Computer Science). School of Computer Science, Carnegie Mellon University. December, 1991. Thesis title: *Programming with Intersection Types and Bounded Polymorphism*. Supervisors: John Reynolds and Robert Harper. M.A. (Computer Science). School of Computer Science, Carnegie Mellon University. 1988. Supervisor: Nico Habermann. B.A. (Linguistics), Stanford University, 1985.

Employment.
Jan 02 to present: Henry Salvatory Professor, Department of Computer and Information Science, University of Pennsylvania. Feb 06 to May 06: Visiting Researcher, Microsoft Research, Cambridge, England. Aug 98 to Dec 01: Associate Professor, Department of Computer and Information Science, University of Pennsylvania. Aug 96 to Jul 98: Assistant Professor, Computer Science Department, Indiana University. Jan 95 to Aug 96: Senior Research Fellow, University of Cambridge, Computer Laboratory. Jan 92 to Dec 94: Research Fellow, University of Edinburgh, Laboratory for Foundations of Computer Science. Sep 92 to May 93: Postdoctoral Fellow, Projet Formel, INRIA-Roquencourt, France. (On leave from LFCS.)

Selected Publications.

1. H. Goldstein,J. W. Cutler, D. Dickstein, B. C. Pierce, and A. Head, "Property-Based Testing in Practice," in *International Conference on Software Engineering (ICSE)*, 2024.
2. H. Goldstein, "Tyche: In situ exploration of random testing effectiveness (demo)," in *ACM Symposium on User Interface Software and Technology (UIST)*, Oct. 2023. at level 9 at level 999
3. H. Goldstein and B. C. Pierce, "Parsing randomness," *Proc. ACM Program. Lang.*, no. OOPSLA, 2022.
4. H. Goldstein, J. W. Cutler, A. Stein, B. C. Pierce, and A. Head, "Some problems with properties," in *Workshop on Human Aspects of Types and Reasoning Assistants (HATRA)*, vol. 1, Dec. 2022.
5. H. Goldstein, J. Hughes, L. Lampropoulos, and B. C. Pierce, "Do judge a test by its cover: Combining combinatorial and property-based testing," in *Programming Languages and Systems, 30th European Symposium on Programming (ESOP)*, 2021.
6. L. Lampropoulos, M. Hicks, and B. C. Pierce, "Coverage guided, property based testing," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, pp. 181:1–181:29, 2019.

Andrew Head · Curriculum Vitae

Assistant Professor · Department of Computer and Information Science · University of Pennsylvania
head@seas.upenn.edu · (610)563-5504 · https://andrewhead.info

Research Interests

Human-computer interaction · Interactive programming tools · Reading tools

Employment

2022–Present	Assistant Professor, University of Pennsylvania
2021	Postdoctoral Scholar, Allen Institute for AI
2020-2021	Postdoctoral Scholar, UC Berkeley

Selected Publications

C.5. Harrison Goldstein, Joseph W. Cutler, Daniel Dickstein, Benjamin C. Pierce, and Andrew Head. Property-Based Testing in Practice. ACM/IEEE ICSE '24. To appear.

C.4. Harrison Goldstein, Benjamin C. Pierce, Andrew Head. Tyche: In Situ Analysis of Random Testing Effectiveness. ACM UIST '23 Demo Track.

C.3. Harrison Goldstein, Joseph W. Cutler, Adam Stein, Benjamin C. Pierce, Andrew Head. Some Problems with Properties. SPLASH HATRA '22.

C.2. Andrew Head, Caitlin Sadowski, Emerson Murphy-Hill, and Andrea Knight. When Not to Comment: Questions and Tradeoffs with API Documentation for C++ Projects. ACM ICSE '18.

C.1. Andrew Head, Elena L. Glassman, Björn Hartmann, and Marti A. Hearst. Interactive Extraction of Examples from Existing Code. ACM CHI '18. *Nominated for Best Paper Award*.

Selected Awards and Recognitions

2019, 2022, 2023	Best of CHI Best Paper Award (top 1% of submissions)

Selected Invited Talks

Interactive Program Distillation. Penn '21, '22, Microsoft Research '21, ASU '20.

Interactive Authoring and Reading with IDEs for Ideas. BayCHI '22, Allen Institute for AI '21, UMich CSE '21, Adobe Research Document Intelligence group '21, Pitt '21, Cornell '21, Penn '21.

Selected Teaching

Fa '22	Lecturer, *Designing Programming Environments: Live and Literate Programming*
Sp '22, '23	Lecturer, *Introduction to Human-Computer Interaction*

Selected Service

Program committee. IEEE VL/HCC '22, '23. ACM UIST '21, '22. SPLASH HATRA '21.

Organizing committee. ACM UIST '24, IEEE VL/HCC '21.