# Brief Description

Human-Centered Programming Languages is the first truly interdisciplinary textbook on programming language design, putting methods from computer science on equal footing with those from the social sciences and humanities. This project is motivated by the observation that design has long been a topic of interest for students and educators in upper-level elective computer science courses on programming language foundations, yet intradisciplinary computer science courses have struggled to provide a foundation for its study. Many programming language textbooks are devoid of design discussions, others contain brief anecdotes, notes, or offhand comments, yet none approaches the topic of design with the same systematic rigor otherwise applied to the study of programming languages. An interdisciplinary framing simultaneously meets this lasting need and helps keep up with the times: by providing multiple disciplinary perspectives on programming languages, it caters to students from diverse intellectual problems and even provides a strong foothold to engage with issues of social inclusion in design.

Interdisciplinary pedagogy presents a unique challenge: most students and educators have strong background in only a single discipline, yet they are expected to engage with several. Human-Centered Programming Languages overcomes this challenge both in content presentation and exercise design. Firstly, content is presented through the lens of five fictional characters (archetypes) representing different scholarly perspectives, helping readers contextualize each viewpoint presented in the text while identifying their own areas of strength and weakness. Secondly, the textbook’s supplementary materials include classroom-tested recommendations for grading schemes that build in a recognition of students backgrounds: the recommend scheme uses a combination of autograded programming assignments with peer-reviewed design assignments reviewed by completion (i.e., points assigned for every problem attempted).

# Outstanding Features

On the surface, Human-Centered Programming Languages’ most recognizable distinguishing feature is the use of archetypes to capture distinct perspectives: the Practitioner (writes application code), Implementer (writes language implementations), the Theorist (writes proofs), the Social Scientist (e.g., studies the experiences of humans), and the Humanist (e.g., performs critiques of texts). This visible feature foreshadows the deeper outstanding feature of the book: the breadth and novelty of the contents.

Breadth and novelty are particularly highlight in the chapters on gender, disability, and play: each shows how diverse intellectual traditions fundamentally impact substantial design decisions. The gender chapter uses a contemporary design method from the human-computer interaction community (GenderMag) to explore gender differences in experience of programming language interfaces. The disability chapter shows how the notion of disability as a spectrum, drawn from disability studies, fundamentally shaped both the physical interface of a language for disabled children and the evaluation thereof. The chapter on play shows that well-trodden theorems such as the pumping lemma, progress, and preservation can be recontextualized to provide surprising design insights in the seemingly-unrelated topic of text adventure games.

# Competition

The largest sets of competing texts come from disciplinary academic textbooks for programming language courses (List 1), followed by books on compilers and interpreters (List 2). Compared to these books, the proposed project represents a trade-off in breadth of coverage vs. depth. The proposed book cannot and does not attempt to complete with List 1 on mathematical depth nor complete with List 2 on the depth of its coverage of implementation, yet its breadth sets it apart: no book on List 1 nor List 2 integrates perspectives from the social sciences nor humanities.

To substantiate my claim to the first truly interdisciplinary textbook on programming languages, I highlight near-neighbors from outside pure computer science in List 3.

Extra-disciplinary books in the social sciences and humanities with clear relevance to programming languages (list 3) are fewer in number. Book 1 has strong foundations in psychology, yet it is about programmers, for programmers, not language design. Moreover, it is a trade book and avoids direct engagement with the research literature, which is a necessity for this emerging topic. Book 2 is about the rhetorical content of code as opposed to language design. Though it is firmly situated in the humanities, its audience likely overlaps with ours. Book 3 is notable because the live programming community is characterized by a longstanding interest in both programming languages and human experience, yet this focus is much too narrow for our purposes.

Though List 3 is short, there is substantial prior writing on the interaction of humans with programming languages in the form of research papers. Notable authors include Amy Ko, Brad Myers, Jonathan Aldrich, Michael Coblenz, Shriram Krishnamurthy, Will Crichton, and Jun Kato. The most notable venue is IEEE Symposium on Visual Languages and Human-Centric Computing, though many relevant publications are in general-purpose software engineering and human-computer interaction venues.

The book’s style incorporates elements from that of Shriram Krishnamurthy’s *Programming Languages: Application and Interpretation,* particularly the use of small inline exercises to check the reader’s understanding during a first read, as well as the central role of implementation of an interpreter. I contrast the style with texts such as *Types and Programming Languages* or *Practical Foundations for Programming Languages,* which each assume relative mathematical maturity, e.g., a strong technical vocabulary. A noteworthy element of *Human-Centered Programming Languages’* style is that it provides bolded definitions and examples of core technical terms which might be introduced in passing by other texts. This style aims to remain accessible to students with less background, without compromising rigor.

List 1:

1. Types and Programming Languages, Benjamin Pierce
2. Advanced Types and Programming Languages, Benjamin Pierce
3. Theories of Programming Languages, John Reynolds
4. Essentials of Programming Languages, Daniel P Friedman, Mitchell Wand, Christopher Thomas Haynes (2001)
5. Practical Foundations for Programming Languages, Robert Harper (2016) 2nd ed.
6. Programming Languages: Application and Interpretation, Shriram Krishnamurthy (2022), 3rd ed.
7. Programming Languages: Principles and Paradigms, Maurizio Gabbrielli and Simone Martini (2010)
8. Concepts of Programming Languages, Robert Sebesta (2015)

List 2:

1. Crafting Interpreters, Robert Nystrom (2021)
2. Essentials of Compilation, Jeremy Siek (2023)
3. Compilers: Principles, Techniques, and Tools, Aho, Ullman, and Sethi (2006), 2nd ed.
4. Build Your Own Programming Language: A programmer's guide to designing compilers, interpreters, and DSLs for solving modern computing problems, Clinton L. Jeffrey (2021)
5. Introduction to Compilers and Language Design, Douglas Thain (2020)
6. CPython Internals: Your Guide to the Python 3 Interpreter, Anthony Shaw (2021)
7. Modern Compiler Implementation in ML, Andrew Appel (2004)
8. Compiling with Continuations Revised, Andrew Appel (2007)
9. Engineering a Compiler, Keith Cooper and Linda Torczon (2022), 3rd ed.
10. A Retargetable C Compiler: Design and Implementation, David Hanson and Christopher Fraser (1995)
11. Compiler Construction: Principles and Practice, Kenneth C. Louden (1997)
12. Advanced Compiler Design and Implementation, Steven Muchnick (1997)

List 3:

1. The Programmer's Brain: What every programmer needs to know about cognition, Felienne JJ Hermans (2021)
2. Rhetorical Code Studies, Kevin Brock (2019)
3. Live Coding: A User’s Manual, Alan Blackwell, Emma Cocker, Geoff Cox, Alex McLean and Thor Magnusson (2022)

# Apparatus

The current draft contains 178 references to the research literature and 138 exercises, or 262 if counting subproblems. It comes with complete slides and exercises. The slides consist of 15 slide decks representing approximately 33 hours of lecture time; they contain numerous illustrations, animations, and prompts for interactions or in-class work. For instructors who prefer to prepare their own lectures, each chapter contains a list of recommendations for classroom activities, totaling 79 recommendations. Full materials are provided for five assignments, each consisting of a programming activity and a written activity. Test cases, starter code, and autograding infrastructure are provided for all programming assignments (in the Rust programming language) and peer-evaluation rubrics are provided for key written assignments.

If selected for publication, I plan additional apparatus. Firstly, I wish to use dialogues between the archetypes to synthesize competing perspectives – currently only three chapters have such dialogues. Secondly, I wish to use cartoon illustrations of the archetypes, approximately one per chapter, to highlight key concepts and promote a sense of welcoming.

# Audience

The primary audience is upper undergraduates (3rd and 4th-year) and junior graduate students in computational majors such as computer science, software engineering, and human-computer interaction. It is written with the assumption that it is students’ first book on programming languages, but balances its accessible tone with the need for rigor through thorough research citations.

The text is more descriptive in nature than quantitative, and teaches rigorous qualitative research methods for performing in-depth language design case studies. Yet, much of what the book describes is analytical or algorithmic in nature: the book presents traditional mathematical formalizations and algorithms for parsing, evaluation, and type-checking. For example, formal judgements and inference rules are presented for a simple operational semantics and static type system.

As a prerequisite, the book assumes the reader knows how to program in multiple programming languages (i.e., it assumes exposure to multiple paradigms). The book is written in a programming-language-agnostic fashion, excepting one chapter dedicated to introducing the Rust programming language. This chapter is included because the author’s course uses Rust; it is intended that other instructors can safely ignore this chapter if they wish to use a different implementation language. For a Rust-based course, prior systems-level (e.g. C) language should be made a prerequisite, as this experience helps contextualize the design of Rust. The book does not assume formal languages and automata theory as prerequisites – relevant background is presented from first principles.

# Authors

Rose Bohrer is an Assistant Professor of Computer Science at Worcester Polytechnic Institute since Fall 2021, and has taught four instances of programming languages courses there at both the graduate and undergraduate level. Her PhD thesis at Carnegie Mellon University was nominated by the school for the ACM Dissertation Award, representing that it was ranked one of the top two theses that year. Her dissertation work focuses on theorem-proving with substantial elements of programming language theory. In more recent years, she has also began to publish human-computer interaction work in venues such as CHI Play, contributing to the interdisciplinary knowledge underlying this project.

The author has written two research papers of approximately 10,000 words each about the pedagogy underlying this book project. The [first](https://users.wpi.edu/~rbohrer/pub/splashe23.pdf) was published in the SPLASH Symposium on Programming Language Education in 2023 (acceptance rate: ~60%) and the second is currently under review for Trends in Functional Programming in Education.

# Market Considerations

## Defining the Target Market

I characterize the target market using the ACM Computer Science curriculum guidelines. Because the target curricular topics are core topics, it is appropriate to assert that CS undergraduate programs in general are the target market.

The ACM Computer Science curriculum [guidelines](https://www.acm.org/education/curricula-recommendations) divide the CS curriculum into two tiers of core curriculum as well as electives. These guidelines recommend 28 credit-hours of core programming languages curriculum, including the topics of functional programming, basic type systems, program representation, and language translation/execution. The Human-Centered Programming Languages book is intended to fit this role within the computer science curriculum, and its intended market consists of core programming languages instruction throughout computer science undergraduate and graduate programs.

In addition to this core curricular function, the book addresses the following programming languages elective topics to differing degrees: Syntax Analysis, Advanced Programming Constructs, Concurrency and Parallelism, Type Systems, Formal Semantics, and Language Pragmatics.

Though human-computer interaction courses are not the primary intended market, the book also integrates core curriculum from the “HCI/User-Centered Design and Testing” and “HCI/Design-Oriented HCI” guidelines, making it suitable for courses cross-listed between CS and HCI programs.

## Market Considerations for Language Choice

Choice of programming language is often a factor in CS textbook adoption decisions. To maximize the potential audience, the book is written in language-agnostic fashion except for one chapter dedicated to the programming language Rust. Programming Languages textbooks have a long tradition of using languages from outside the top few, such as Haskell, the Lisp family, and the ML (e.g. OCaml) family. This book uses the Rust programming language because its design provides much to say about the foundations of programming languages and because its popularity is growing. According to [TIOBE](https://www.tiobe.com/tiobe-index/rust/), usage has doubled in the past 18 months and is currently triple that of Haskell, ML, or Lisp, indicating a larger market than existing texts which use those languages.

## Early Adoption Statistics

The first public draft of *Human-Centered Programming Languages* was publicly announced on October 25, 2023 – the announcement received 13,000 views on Twitter. As of December 7th, metrics report 864 unique users and 1,528 page views for the book website itself. During this time, the author used the draft in two courses totaling 83 students – the publishing platform does not distinguish views by students and non-students. Prior to this release, the author created a semi-public Discord server on July 9th 2023 to distribute and discuss an early draft. The server attracted 61 members at that time.

## Promotion Considerations

The scope of the book intersects with the following ACM special interest groups: SIGPLAN, SIGSOFT, SIGCHI, and thus venues and mailing lists associated with these groups may be appropriate for promotion. The audience of the conference VL/HCC is likely to show particular interest. The TYPES mailing list is also of interest.

In an early sign of word-of-mouth promotion, I observed a senior scholar (Neel Krishnaswamy) independently spread news of the book in a 149-member professional Slack group to which we both belong.

# Status

A complete manuscript has been written, totaling about 109,000 words. Notes which later formed the basis of the manuscript were used in a graduate course of approximately 18 students in Spring 2023, then the complete manuscript was used in two courses totaling 83 students (undergraduate and graduate) in Fall 2023. The manuscript has been revised based on student experience.

The manuscript features 78 code listings, 3 figures, and 3 tables. A major goal of future revisions is to increase the figure count to approximately 50, drawing inspiration from illustrations in the slides.

Beyond the current manuscript, I intend to add one chapter covering cross-cultural considerations in programming language design, drawing inspiration from the papers of Felienne Hermans. This addition as well as the addition of further figures, exercises, worked examples, and citations are the main tasks I foresee before an initial manuscript submission. These additions can be made within a time span of four months at a comfortable working pace.

# Reviewers

I recommend the following reviewers based on the literature cited and on my professional network:

* Amy J. Ko
* Jonathan Aldrich
* Michael Coblenz
* Will Crichton
* Shriram Krishnamurthy
* Felienne JJ Hermans
* Jun Kato
* Brad Myers
* Chris Martens
* Ronald Garcia
* Neel Krishnaswamy
* Joseph Osbourne
* Sam Tobin-Hochstadt
* Joshua Grochow
* Yao Li
* Cyrus Omar
* Matt Teichman
* Taylor Smith (StFX)
* Adolfo Neto
* Dan Licata
* Kristopher Micinski