Towards Human-Centered Types & Type Debugging

Kasra Ferdowsi [1]

[1] *University of California, San Diego, CA, USA*

Abstract

Advanced type systems, such as Rust's Ownership, are gaining wider popularity among mainstream programming languages. And yet, despite users' evident struggles with working with these systems, research on tools and techniques for improving the usability of types is rather scarce, and focused mostly on improving type error messages. In this paper, I hope to spark a discussion on human-centered tools and techniques for working with advanced type systems by surveying previous works and synthesizing them into a set of sketches for future research.

Keywords: Static Semantics. Type Errors. Human-Centered Type Debugging.

1 Introduction

Programming Language researchers have spent decades developing type systems with a variety of crucial benefits, from the most basic static guarantees about program behavior, to safe concurrency [1] and automatic memory management without garbage collection [2]. And we have finally begun to see such systems influencing mainstream programming languages [3]. Rust, with its combination of affine and region types, is slowly but surely gaining in popularity [4]. TypeScript seems to be overtaking JavaScript in some areas [5]. Even Python, which long held out as a dynamically typed language, has added a powerful type system, a strict static type checker pyright, and features that until recently were only available in functional languages, from union types [6] to variadic generics [7]. Yet, as we have seen from the slow growth of Rust [8], and attestations of the difficulty of debugging type errors in PL research [9]–[16], languages with complex type systems tend to have a steep learning curve that remains a barrier to adopting them.

This presents a clear need for tools and techniques for learning to reason about such advanced type systems[1]. But the history of usability research into such systems is almost entirely restricted to improving type *error messages*, and except for some recent works on the usability of Rust, tended not to include a human-centered evaluation at all. So in this paper, I hope to present a discussion on human-centered design and evaluation of tools for learning and working with advanced type systems by surveying existing works and synthesizing them into a set of sketches for future research.

The rest of the paper is structured as follows: Sec. 2 provides a brief summary of research I am aware of on improving the usability of type systems, and discusses some of the limitations to the approaches taken in this area. In parallel to it, Sec. 3 introduces more recent works on the usability of the Rust programming language which, using human-centered methods, shows some of the real-world challenges faced by users of Rust's infamous Ownership system. Sec. 4 then brings these parallel discussions together and presents sketches for truly human-centered approaches to working with advanced type systems.

2 The Usability of Advanced Type Systems

Research on types has existed for longer than Computer Science itself [17]. Yet, despite a plethora of type systems and features from Linear [1] to Dependent types [18], there has been a general disinterest in the usability aspects of working with such type systems [19].

2.1 Expressiveness vs. Usability

Papers that introduce a type system tend to report what it is like to program using that system. However, rather than considering this an issue of *usability*, they focus on *expressiveness*, i.e. the possibility of writing interesting programs that type check.

PLATEAU
13th Annual Workshop at the Intersection of PL and HCI

DOI: 10.35699/1983-3652.yyyy.nnnnn

Organizers:
Sarah Chasins, Elena Glassman, and Joshua Sunshine

1 A better term would be *static semantics*, both because this argument is not restricted to type systems, and because "What is a type?" is a sure way to start a fight among PL researchers. However, given that the majority of works cited here focus on type errors, and for the sake of brevity, I will use "type" in this paper.

As an example consider Ownership types [20]. The various Ownership type systems grew out of Object-Oriented Programming research, and tended to focus on the popular Java programming language, with many papers evaluating their type system's expressiveness [21]–[24], and some even explicitly investigating their effects on program comprehension [20], [21]. In their evaluations, the authors of these papers only discussed their own experience writing complex programs using the system, implying that their ability to do so in indicative of the usability of the type system generally.

However, as Hage [25] also points out, in reality expressiveness can *oppose* usability, with more expressive type systems causing type errors to become an unreadable "essay of types" [26], and hinder user productivity. This was likely an issue with methodology, since at the time human-centered evaluations such as user studies were uncommon and not a standard the PL community held itself to. Fortunately, as I will discuss in Sec. 3, this seems to be improving as we employ HCI techniques in PL research generally, and on the usability of type systems specifically.

2.2 Human- vs. Tool-Centric Type Debugging

Of course, the PL community has not totally ignored this usability issue with type systems. As mentioned in Sec. 1, researchers have been aware of the issues with type errors for some time, resulting in a number of works on type error debugging. These include works on Type Error Slicing [27], [28], Counter-Factual Typing [29]–[31] and related type systems [12], [32]–[35] which have tried to address the specific challenges novice Functional programmers face [11]. These works tend to focus on improving type error messages, with many specifically addressing the problems of error *localization* and *correction*, i.e. which subexpression causes the type error, and what local edit will fix it[2].

I posit that these works, while containing great technical contributions, have a key limitation that has affected their real-world applicability: Rather than a *human-centric* approach which focuses on helping the user understand and fix the error, they center their *tool* and assume that the solution is a purely technical one of finding a more sophisticated static analysis.

For example, consider Guided Type Debugging [12]. This work improves upon previous approaches [10], [36] which employed the user as an oracle for an automated type debugger: by asking for the intended type of various subexpressions from the user, the type debugger can localize the type error, and suggest an edit for fixing it. Notice that in this work, even as they bring the user in-the-loop of the type debugging process, their focus is on the *tool*'s ability to localize and correct the error, not the *human*'s understanding of it. And they consider this human involvement at all a downside [37].

Due to the lack of both user evaluation in these papers and wider adoption of these techniques, it remains to be seen if and how these technical contributions improve the usability of advanced type systems. In recent years, however, a different line of research has employed human-centered methods to evaluate the usability of Rust's Ownership semantics. So before returning to how we may address the limitations of the research surveyed in this section, I will summarize some key findings on working with advanced type systems that we have learned from the work on Rust.

3 The Usability of Rust's Ownership

With the emergence of human-centered methods in Programming Languages research [38], we have seen a small but significant recent set of works that actually evaluate a program's advanced type system with non-experts. Specifically, for the Ownership system of the Rust programming language. A thorough review of Rust's Ownership is outside the scope of this paper, but in short Rust combines ideas from Linear [1] and Region [39] types[3] to statically guarantee at-most single mutable access to each value at any point in the program. This guarantee is enforced by a pass in the compiler colloquially known as the *borrow checker*. As one may expect, experience and research show that Rust's Ownership is difficult to learn, and suggest that the difference in *paradigm* enforced by the type system, and the complicated implementation of Ownership rules, are arguably more significant factors

2 A deeper discussion of these works in beyond the scope of this paper. See Hage [25] for a great survey.

3 Frustratingly, despite its name, Rust's Ownership is not directly related to Ownership types, though it shares some of their ideas. And, to be accurate, Rust employs Affine [40], not Linear, types. See Ferdowsi [41] for a more thorough discussion of Rust's lineage.

than the quality and accuracy of error messages.

3.1 High-Level Architectural Changes

A common thread among various research on Rust novices is that the biggest challenge in learning Rust is that Ownership is a very different programming paradigm [42] and, to quote an experienced C++ developer, "forces the programmer to think differently." [43]. More concretely, this change in paradigm requires not just understanding the rules of the borrow checker, but also how these rules require small and large changes to the structure of the code itself.

We can see this most clearly in Coblenz et. al. [44]. In their study on whether Bronze, a garbage-collected wrapper for Rust, improved students' performance in completing a programming assignment, they found that "most of the benefit of GC comes from architectural simplification" and that "design was a significant contributor to the difference in performance between non-Bronze and Bronze participants". And in Zeng and Crichton [8] who found that "the complexity of the borrow checker was the second most frequently mentioned complaint"[4] in online experience reports they inspected. Specifically, the rules of Ownership disallow many memory access patterns that are common in other languages, which leads to frustration.

3.2 Low-Level Minutiae of the Borrow Checker

But the challenges in learning Rust's Ownership were not restricted to its implications for how to structure the code. At the other end of the scale, some research suggests that the particular minutiae of the *implementation* of the Ownership rules are also a notable challenge for Rust programmers. This point is made most concisely by Crichton [45], who argues that the rules of Ownership are simple, but using Rust effectively requires an understanding, not just of these rules, but of precisely how they are implemented in the borrow checker.

Empirically, this is backed up by Qin et. al. [46], who found that the subtleties of the borrow checker's type inference made writing block-free thread-safe code difficult. This is particularly notable, as it indicates how even experienced Rust developers still struggle with "the vagaries of the borrow checker" [45]. And demonstrates the need for inspecting, in detail, how the borrow checker reasons about the user's code locally.

3.3 Beyond Error Message Accuracy

It's worth mentioning here that neither the high-level, nor the very low-level usability challenges in Rust's Ownership can be easily addressed by improving error messages alone. Rust has famously well-designed error messages [42] that, while imperfect [47], seem to follow all the best guidelines [48]. And error localization and correction, the focus of many previous works, don't seem to be an issue with the Rust compiler.

Instead, Coblenz et al. [44] found that Rust error messages are unhelpful for two reasons. The first is that they discourage the user from engaging with the type error by providing (a series of) edit suggestions that can fix the errors without helping the user understand and learn from their error. The second, as mentioned above, is that the local type errors may signify *architectural* problems which cannot be addressed by local fixes at all. In other words, what users need is not accurate localization and correction, but feedback that "aid design or comprehension."

We must be careful here not to overgeneralize from these findings, as Rust's Ownership is arguably more strict and a more radically different paradigm than, for instance, supporting dependent types. But we argue that this research can nevertheless shed some light on why it and other advanced type systems failed to find solid footing in mainstream languages, and how we may best aid programmers in adopting and working with them now.

4 Second only to compiler version issues.

4 Sketches for Human-Centered Types and Type Debugging

Given the limitations discussed in Sec. 2, and the empirical findings on the usability of Rust's Ownership from Sec. 3, how can we design truly human-centered tools and techniques for working with advanced type systems? The real answer is of course unknown, so rather than definitive "guidelines" or "paradigms", I will conclude here by offering sketches, grounded in existing works, of what these tools and techniques may be like. I hope that these sketches will encourage and facilitate further research and discussions on human-centered types and type debugging[5].

4.1 Type Exploration

One need demonstrated by the research on Rust, particularly Sec. 3.2, is the ability to inspect how the type checker reasons about particular parts of a user's code, regardless of the existence of type errors. And one solution to this is tools for visualizing and exploring the type checker's reasoning.

Some works for Rust have already attempted to do this, though they struggle with the size and complexity of the borrow checker's reasoning. Dominik [52] and Blaser [53] directly visualize the set of constraints leading to a type error, which result in large graphs that are difficult to follow. And RustViz [54] avoids this by placing the burden of writing educational code and appropriate levels of detail in the visualization to a human teacher.

Instead, we may overcome this by following Scalad [55]–[57] which presents an interactive view of the Scala Type Derivation Tree, allowing the programmer to explore how the type checker reasons about their code. This work also lacks a user evaluation [58], so its usability is not clear, but I argue that it is a promising framework for approaching human-centered type exploration systems.

Rather than directly visualizing the type derivation rules, it maps them to a "high-level representation" aimed at answering users' questions. While I suspect that Scalad's visualizations are still too complex, this approach can be extended by designing and evaluating alternative representations targeted at improving users' mental models[6]. And by being designed around interactivity, it allows for exploring different ways of *engagement* with the visualizations, which Sorva [61] argues is more important for learning than the quality of visualizations. Finally, as this approach can work with arbitrary code, it is specifically suited to how experienced developers learn a new programming language [43]: by programming in that language, and learning opportunisitcally as they run into issues with their own code.

4.2 Human-Centered Type Debugging

Another valuable future direction would be to continue the works focused on interactive type debugging. But, as I argued in Sec. 3.1, an important aspect to type debugging is fixing the broader architecture of the code, not just local type errors. And until we find technical solutions that can identify and fix such errors automatically, we need a way to assist the programmer themself in distinguishing local and architectural errors, and fixing them accordingly.

One possible technique for this comes from Algorithmic Debugging of Type Errors [10], an interaction model which presents the unsatisfiable type constraint at the location of the error to the user, and allows them to ask for an explanation of any of the types at that location. They are then guided through their code as they identify the source of the error. Chen and Erwig [12] criticized the number of steps required to fix the bug in this approach. But by centering the user in the debugging process, and allowing them to explore their code from the location of the reported error to its origin elsewhere, this model may be able to bridge the gap between local types and their wider architectural implications, providing the much needed aid in design and comprehension.

5 This section is necessarily speculative and limited. Other important discussions include standardized methods for evaluating human-centered tools and techniques (e.g. Paradigm Problems [49]), addressing the different classes of type errors (similar to static analysis [50]), accounting for the spectrum of users' general programming expertise and expertise in a specific type system, dealing with bugs in type checkers themselves [51], and of course remaining technical challenges [25].

6 A discussion of specific techniques and theories for this design is outside the scope of this paper, though an understanding of Notional Machines [59], [60] may be a good place to start.

References

[1] P. Wadler, "Linear types can change the world!" In *Programming Concepts and Methods*, 1990.

[2] M. Tofte and J.-P. Talpin, "Region-based memory management," *Information and Computation*, vol. 132, no. 2, pp. 109–176, 1997, ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.1996.2613. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540196926139.

[3] N. Savage, "Gradual evolution," *Commun. ACM*, vol. 57, no. 10, pp. 16–18, Sep. 2014, ISSN: 0001-0782. DOI: 10.1145/2659764. [Online]. Available: https://doi.org/10.1145/2659764.

[4] Stack Overflow, *2022 Developer Survey*, en, 2022. [Online]. Available: https://survey.stackoverflow.co/2022/ (visited on 11/22/2022).

[5] R. Powell, *The 2022 state of software delivery*, en, 2022. [Online]. Available: https://circleci.com/resources/2022-state-of-software-delivery/ (visited on 11/22/2022).

[6] P. Galindo, *Python 3.10.0 is available*, en, 2021. [Online]. Available: https://blog.python.org/2021/10/python-3100-is-available.html (visited on 11/22/2022).

[7] P. Galindo, *Python 3.11.0 is now available*, en, 2022. [Online]. Available: https://blog.python.org/2022/10/python-3110-is-now-available.html (visited on 11/22/2022).

[8] A. Zeng and W. Crichton, "Identifying barriers to adoption for rust through online discourse," in *Proceedings of the 9th Workshop on Evaluation and Usability of Programming Languages and Tools, 2018*, ser. PLATEAU '18, 2018. DOI: 10.48550/arxiv.1901.01001. [Online]. Available: https://arxiv.org/abs/1901.01001.

[9] C. Clack and C. Myers, "The dys-functional student," in *Funtional Programming Languages in Education*, P. H. Hartel and R. Plasmeijer, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 289–309, ISBN: 978-3-540-49252-8.

[10] O. Chitil, "Compositional explanation of types and algorithmic debugging of type errors," in *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '01, Florence, Italy: Association for Computing Machinery, 2001, pp. 193–204, ISBN: 1581134150. DOI: 10.1145/507635.507659. [Online]. Available: https://doi.org/10.1145/507635.507659.

[11] B. S. Lerner, M. Flower, D. Grossman, and C. Chambers, "Searching for type-error messages," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07, San Diego, California, USA: Association for Computing Machinery, 2007, pp. 425–434, ISBN: 9781595936332. DOI: 10.1145/1250734.1250783. [Online]. Available: https://doi.org/10.1145/1250734.1250783.

[12] S. Chen and M. Erwig, "Guided type debugging," in *Functional and Logic Programming*, M. Codish and E. Sumii, Eds., Cham: Springer International Publishing, 2014, pp. 35–51, ISBN: 978-3-319-07151-0.

[13] D. Zhang and A. C. Myers, "Toward general diagnosis of static errors," in *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '14, San Diego, California, USA: Association for Computing Machinery, 2014, pp. 569–581, ISBN: 9781450325448. DOI: 10.1145/2535838.2535870. [Online]. Available: https://doi.org/10.1145/2535838.2535870.

[14] D. Zhang, A. C. Myers, D. Vytiniotis, and S. Peyton-Jones, "Diagnosing type errors with class," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '15, Portland, OR, USA: Association for Computing Machinery, 2015, pp. 12–21, ISBN: 9781450334686. DOI: 10.1145/2737924.2738009. [Online]. Available: https://doi.org/10.1145/2737924.2738009.

[15] V. TIRRONEN, S. UUSI-MÄKELÄ, and V. ISOMÖTTÖNEN, "Understanding beginners' mistakes with haskell," *Journal of Functional Programming*, vol. 25, e11, 2015. DOI: 10.1017/S0956796815000179.

[16] D. Zhang, A. C. Myers, D. Vytiniotis, and S. Peyton-Jones, "Sherrloc: A static holistic error locator," *ACM Trans. Program. Lang. Syst.*, vol. 39, no. 4, Aug. 2017, ISSN: 0164-0925. DOI: 10.1145/3121137. [Online]. Available: https://doi.org/10.1145/3121137.

[17] B. C. Pierce, "Introduction," in *Types and Programming Languages*. The MIT Press, 2002, pp. 1–13, ISBN: 9780262256810.

[18] H. Xi and F. Pfenning, "Dependent types in practical programming," in *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '99, San Antonio, Texas, USA: Association for Computing Machinery, 1999, pp. 214–227, ISBN: 1581130953. DOI: 10.1145/292540.292560. [Online]. Available: https://doi.org/10.1145/292540.292560.

[19] M. Coblenz, J. Aldrich, B. A. Myers, and J. Sunshine, "Interdisciplinary programming language design," in *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, ser. Onward! 2018, Boston, MA, USA: Association for Computing Machinery, 2018, pp. 133–146, ISBN: 9781450360319. DOI: 10.1145/3276954.3276965. [Online]. Available: https://doi.org/10.1145/3276954.3276965.

[20] D. Clarke, J. Östlund, I. Sergey, and T. Wrigstad, "Ownership Types: A Survey," en, in *Aliasing in Object-Oriented Programming. Types, Analysis and Verification*, ser. Lecture Notes in Computer Science, D. Clarke, J. Noble, and T. Wrigstad, Eds., Berlin, Heidelberg: Springer, 2013, pp. 15–58, ISBN: 978-3-642-36946-9. DOI: 10.1007/978-3-642-36946-9_3. [Online]. Available: https://doi.org/10.1007/978-3-642-36946-9_3 (visited on 03/01/2022).

[21] J. Aldrich, V. Kostadinov, and C. Chambers, "Alias annotations for program understanding," in *Proceedings of the 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '02, Seattle, Washington, USA: Association for Computing Machinery, 2002, pp. 311–330, ISBN: 1581134711. DOI: 10.1145/582419.582448. [Online]. Available: https://doi.org/10.1145/582419.582448.

[22] C. Boyapati, A. Salcianu, W. Beebee, and M. Rinard, "Ownership types for safe region-based memory management in real-time java," in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, ser. PLDI '03, San Diego, California, USA: Association for Computing Machinery, 2003, pp. 324–337, ISBN: 1581136625. DOI: 10.1145/781131.781168. [Online]. Available: https://doi.org/10.1145/781131.781168.

[23] C. Boyapati, B. Liskov, and L. Shrira, "Ownership types for object encapsulation," in *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '03, New Orleans, Louisiana, USA: Association for Computing Machinery, 2003, pp. 213–223, ISBN: 1581136285. DOI: 10.1145/604131.604156. [Online]. Available: https://doi.org/10.1145/604131.604156.

[24] D. G. Clarke, J. M. Potter, and J. Noble, "Ownership types for flexible alias protection," in *Proceedings of the 13th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '98, Vancouver, British Columbia, Canada: Association for Computing Machinery, 1998, pp. 48–64, ISBN: 1581130058. DOI: 10.1145/286936.286947. [Online]. Available: https://doi.org/10.1145/286936.286947.

[25] J. Hage, "Solved and open problems in type error diagnosis?" English, *CEUR Workshop Proceedings*, vol. 2707, pp. 62–74, Oct. 2020, Publisher Copyright: © 2020 CEUR-WS. All rights reserved.; 4th Workshop on Model-Driven Engineering for the Internet-of-Things, 1st International Workshop on Modeling Smart Cities, and 5th International Workshop on Open and Original Problems in Software Language Engineering, STAF-WS 2020 ; Conference date: 22-06-2020 Through 26-06-2020, ISSN: 1613-0073.

[26] G. Gjyshinca, *Monad i love you now get out of my type system*, en, Oct. 2022. [Online]. Available: https://www.youtube.com/watch?v=2PxsyWqZ5dI (visited on 01/02/2023).

[27] C. Haack and J. B. Wells, "Type error slicing in implicitly typed higher-order languages," in *Programming Languages and Systems*, P. Degano, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 284–301, ISBN: 978-3-540-36575-4.

[28] T. Schilling, "Constraint-free type error slicing," in *Trends in Functional Programming*, R. Peña and R. Page, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–16, ISBN: 978-3-642-32037-8.

[29] S. Chen and M. Erwig, "Counter-factual typing for debugging type errors," in *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '14, San Diego, California, USA: Association for Computing Machinery, 2014, pp. 583–594, ISBN: 9781450325448. DOI: 10.1145/2535838.2535863. [Online]. Available: https://doi.org/10.1145/2535838.2535863.

[30] S. Chen and B. Wu, "Efficient counter-factual type error debugging," *Science of Computer Programming*, vol. 200, p. 102 544, 2020, ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2020.102544. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642320301520.

[31] K. Tsushima, O. Chitil, and J. Sharrad, "Type debugging with counter-factual type error messages using an existing type checker," in *Proceedings of the 31st Symposium on Implementation and Application of Functional Languages*, ser. IFL '19, Singapore, Singapore: Association for Computing Machinery, 2021, ISBN: 9781450375627. DOI: 10.1145/3412932.3412939. [Online]. Available: https://doi.org/10.1145/3412932.3412939.

[32] T. Jim, "What are principal typings and what are they good for?" In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '96, St. Petersburg Beach, Florida, USA: Association for Computing Machinery, 1996, pp. 42–53, ISBN: 0897917693. DOI: 10.1145/237721.237728. [Online]. Available: https://doi.org/10.1145/237721.237728.

[33] M. Neubauer and P. Thiemann, "Discriminative sum types locate the source of type errors," in *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '03, Uppsala, Sweden: Association for Computing Machinery, 2003, pp. 15–26, ISBN: 1581137567. DOI: 10.1145/944705.944708. [Online]. Available: https://doi.org/10.1145/944705.944708.

[34] S. Chen, M. Erwig, and E. Walkingshaw, "An error-tolerant type system for variational lambda calculus," in *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '12, Copenhagen, Denmark: Association for Computing Machinery, 2012, pp. 29–40, ISBN: 9781450310543. DOI: 10.1145/2364527.2364535. [Online]. Available: https://doi.org/10.1145/2364527.2364535.

[35] S. Chen, M. Erwig, and E. Walkingshaw, "Extending type inference to variational programs," *ACM Trans. Program. Lang. Syst.*, vol. 36, no. 1, Mar. 2014, ISSN: 0164-0925. DOI: 10.1145/2518190. [Online]. Available: https://doi.org/10.1145/2518190.

[36] P. J. Stuckey, M. Sulzmann, and J. Wazny, "Interactive type debugging in haskell," in *Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell*, ser. Haskell '03, Uppsala, Sweden: Association for Computing Machinery, 2003, pp. 72–83, ISBN: 1581137583. DOI: 10.1145/871895.871903. [Online]. Available: https://doi.org/10.1145/871895.871903.

[37] S. Chen and M. Erwig, "Systematic identification and communication of type errors," *Journal of Functional Programming*, vol. 28, e2, 2018. DOI: 10.1017/S095679681700020X.

[38] S. E. Chasins, E. L. Glassman, and J. Sunshine, "Pl and hci: Better together," *Commun. ACM*, vol. 64, no. 8, pp. 98–106, Jun. 2021, ISSN: 0001-0782. DOI: 10.1145/3469279. [Online]. Available: https://doi.org/10.1145/3469279.

[39] D. Grossman, G. Morrisett, T. Jim, M. Hicks, Y. Wang, and J. Cheney, "Region-based memory management in cyclone," *ACM SIGPLAN Notices*, vol. 37, no. 5, pp. 282–293, May 2002, ISSN: 0362-1340. DOI: 10.1145/543552.512563. [Online]. Available: https://doi.org/10.1145/543552.512563 (visited on 02/27/2022).

[40] B. C. Pierce, *Advanced Topics in Types and Programming Languages*. The MIT Press, 2004, ISBN: 0262162288.

[41] K. Ferdowsi, *The usability of advanced type systems: Rust as a case study*, 2022. DOI: 10.48550/arxiv.2301.02308. [Online]. Available: https://arxiv.org/abs/2301.02308.

[42] K. R. Fulton, A. Chan, D. Votipka, M. Hicks, and M. L. Mazurek, "Benefits and drawbacks of adopting a secure programming language: Rust as a case study," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, 2021, pp. 597–616.

[43] N. Shrestha, C. Botta, T. Barik, and C. Parnin, "Here we go again: Why is it difficult for developers to learn another programming language?" In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20, Seoul, South Korea: Association for Computing Machinery, 2020, pp. 691–701, ISBN: 9781450371216. DOI: 10.1145/3377811.3380352. [Online]. Available: https://doi.org/10.1145/3377811.3380352.

[44] M. Coblenz, M. L. Mazurek, and M. Hicks, "Garbage collection makes rust easier to use: A randomized controlled trial of the bronze garbage collector," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 1021–1032, ISBN: 9781450392211. DOI: 10.1145/3510003.3510107. [Online]. Available: https://doi.org/10.1145/3510003.3510107.

[45] W. Crichton, "The usability of ownership," HATRA '20, Nov. 2020. DOI: 10.48550/arxiv.2011.06171. [Online]. Available: https://arxiv.org/abs/2011.06171.

[46] B. Qin, Y. Chen, Z. Yu, L. Song, and Y. Zhang, "Understanding memory and thread safety practices and issues in real-world rust programs," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020, London, UK: Association for Computing Machinery, 2020, pp. 763–779, ISBN: 9781450376136. DOI: 10.1145/3385412.3386036. [Online]. Available: https://doi.org/10.1145/3385412.3386036.

[47] S. Zhu, Z. Zhang, B. Qin, A. Xiong, and L. Song, "Learning and programming challenges of rust: A mixed-methods study," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 1269–1281, ISBN: 9781450392211. DOI: 10.1145/3510003.3510164. [Online]. Available: https://doi.org/10.1145/3510003.3510164.

[48] B. A. Becker, P. Denny, R. Pettit, *et al.*, "Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research," in *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '19, New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 177–210, ISBN: 978-1-4503-7567-2. DOI: 10.1145/3344429.3372508. [Online]. Available: https://doi.org/10.1145/3344429.3372508 (visited on 02/27/2022).

[49] W. Crichton, *Paradigm problems: A case study on rebalance*, en, Sep. 2022. [Online]. Available: https://www.youtube.com/watch?v=zsC6VdC28Bc (visited on 01/04/2023).

[50] Y. P. Khoo, J. S. Foster, M. Hicks, and V. Sazawal, "Path projection for user-centered static analysis tools," in *Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, ser. PASTE '08, Atlanta, Georgia: Association for Computing Machinery, 2008, pp. 57–63, ISBN: 9781605583822. DOI: 10.1145/1512475.1512488. [Online]. Available: https://doi.org/10.1145/1512475.1512488.

[51] S. Chaliasos, T. Sotiropoulos, G.-P. Drosos, C. Mitropoulos, D. Mitropoulos, and D. Spinellis, "Well-typed programs can go wrong: A study of typing-related bugs in jvm compilers," *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, Oct. 2021. DOI: 10.1145/3485500. [Online]. Available: https://doi.org/10.1145/3485500.

[52] D. Dominik, "Visualization of Lifetime Constraints in Rust," en, ETH Zurich, Dec. 2018. [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/infk/chair-program-method/pm/documents/Education/Theses/Dominik_Dietler_BA_report.pdf (visited on 03/01/2022).

[53] D. Blaser, "Simple Explanation of Complex Lifetime Errors in Rust," en, ETH Zurich, Aug. 2019. [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/infk/chair-program-method/pm/documents/Education/Theses/David_Blaser_BA_Report.pdf (visited on 03/01/2022).

[54] M. Almeida, G. Cole, K. Du, *et al.*, "Rustviz: Interactively visualizing ownership and borrowing," in *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2022, pp. 1–10. DOI: 10.1109/VL/HCC53370.2022.9833121.

[55] H. Plociniczak and M. Odersky, "Implementing a type debugger for scala," 2012. [Online]. Available: http://infoscience.epfl.ch/record/179877.

[56] H. Plociniczak, "Scalad: An interactive type-level debugger," in *Proceedings of the 4th Workshop on Scala*, ser. SCALA '13, Montpellier, France: Association for Computing Machinery, 2013, ISBN: 9781450320641. DOI: 10.1145/2489837.2489845. [Online]. Available: https://doi.org/10.1145/2489837.2489845.

[57] H. Plociniczak, H. Miller, and M. Odersky, "Improving human-compiler interaction through customizable type feedback," 2014. [Online]. Available: http://infoscience.epfl.ch/record/197948.

[58] H. Plociniczak, "Decrypting local type inference," p. 337, 2016. DOI: 10.5075/epfl-thesis-6741. [Online]. Available: http://infoscience.epfl.ch/record/214757.

[59] J. Sorva, "Notional machines and introductory programming education," *ACM Transactions on Computing Education*, vol. 13, no. 2, 8:1–8:31, Jul. 2013. DOI: 10.1145/2483710.2483713. [Online]. Available: https://doi.org/10.1145/2483710.2483713 (visited on 05/20/2021).

[60] P. E. Dickson, N. C. C. Brown, and B. A. Becker, "Engage against the machine: Rise of the notional machines as effective pedagogical devices," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '20, Trondheim, Norway: Association for Computing Machinery, 2020, pp. 159–165, ISBN: 9781450368742. DOI: 10.1145/3341525.3387404. [Online]. Available: https://doi.org/10.1145/3341525.3387404.

[61] J. Sorva, V. Karavirta, and L. Malmi, "A Review of Generic Program Visualization Systems for Introductory Programming Education," en, *ACM Transactions on Computing Education*, vol. 13, no. 4, pp. 1–64, Nov. 2013, ISSN: 1946-6226. DOI: 10.1145/2490822. [Online]. Available: https://dl.acm.org/doi/10.1145/2490822 (visited on 05/20/2021).