# “Against the Void” - Appendix

## Contents

# 1 Interviews

## 1.1 Screening Survey

### 1.1.1 Eligibility

1. [ *Required for eligibility, yes/no* ] Do you have more than 1 year of experience using Rust?
2. [ *Required for eligibility, yes/no* ] Do you regularly reason about the correctness of unsafe code?
3. [ *Required for eligibility if no to previous, yes/no* ] Do you regularly write new Rust code or edit existing Rust code within an `unsafe` block or a function?

### 1.1.2 Background

1. [ *Required ≥1 for eligibility, short answer* ] How many years of experience do you have using Rust?
2. [ *yes/no* ] Do you call foreign functions from Rust

   [ *if **yes*** ]

   2.1. [ *check all languages that apply* ] Foreign functions that you call ***from*** Rust are written in:
3. [ *yes/no* ] Do you call Rust functions from other languages?

   [ *if **yes*** ]

   3.1. [ *check all languages that apply* ] Your code that calls ***foreign Rust*** functions is written in:
4. [ *if **yes** to 2 or 3, select all that apply* ] Select the tools and methods you use to create foreign bindings.
   - Manually written, bindgen, cbindgen, wasm, bindgen, cxx, autocxx, PyO3, hs-bindgen, Other (short answer)
5. [ *yes/no* ] Do you use Rust's intrinsics?
6. [ *yes/no* ] Do you use system calls?
7. [ *select all that apply* ] Which of the following reference and memory container types have you converted to raw pointers or used to contain raw pointers?
   - `Box`, `Rc`, `UnsafeCell`, `Cell`, `Arc`, `RefCell`, `MaybeUninit`, `ManuallyDrop`, Other (short answer)
8. [ *select all that apply* ] Which of the following bug-finding tools do you use with codebases containing `unsafe` Rust?
   - Clippy, Miri, AddressSanitizer (ASAN), ThreadSanitizer (TSAN), UndefinedBehaviorSanitizer (UBSAN), MemorySanitizer (MSAN), Other (short answer)

### 1.1.3 Personal Information

The following questions request identifying information. Your affiliation will be collected so that we can determine if experiences with unsafe Rust vary based on the institutional context where development takes place. The specifics of your affiliation will not be published. Your affiliation will only be referred to in published data under broad classifications similar but not limited to "industry," "academia," and "open-source." Your name and email are collected for scheduling and payment purposes.

1. [ *short answer* ] What is your name?
2. [ *short answer* ] What is your email?
3. [ *short answer* ] What is your current affiliation?

### 1.2 Interview Protocol

1. What have been your motivations for learning Rust and choosing to use it?
2. What do you use `unsafe` Rust for?
3. You mentioned using [ *list types from screening* ] in `unsafe` contexts. What do you use them for? Describe how you reason about memory safety in these situations.
4. You mentioned using [ *list methods from screening* ] to create foreign bindings. Describe your experiences with these methods.
5. How do you navigate the differences between Rust's memory model and other memory models?
6. Describe a bug you faced that involved `unsafe` Rust.
7. You mentioned using [ *list tools from screening* ]. Describe your experience using each one.
8. Do your development tools handle all of the problems that you face writing `unsafe` Rust?

## 2 Interrater Reliability

We used interrater reliability as a mechanism for refining our codebook. This took place across 7 "rounds" between the first and second authors. During each round, we coded sets of quotes that were randomly selected from across all interviews. To sample quotes, we concatenated all of the interview transcripts into a single document and randomly sampled character indices. The first author visited each index and selected a snippet of text around that location. Each round had a unique set of quotes. When an index was randomly selected and it was near a quote used in a previous round, the first author would choose an adjacent quote or continue sampling to find a new one.

In each of the first 5 rounds, we coded 20 randomly selected quotes. Since our reliability ratings were inconsistent between each round, we switched to coding a larger sample of 60 quotes for rounds 6 and 7. After round 6, we excluded certain themes where establishing reliability was less crucial, since their constituent codes were either keywords in the transcripts ("Interoperation," "Types," "Tools"); corresponded to a singular, abstract theme ("Uncertainty"); or had not been encountered during previous rounds ("Challenges with Bug-finding"). The theme "`unsafe` data structure" was excluded from this round because its purpose was to identify data structures that interview participants had mentioned so that the authors could create prototypical examples for future research, so it was not relevant to our research questions. We also chose smaller quote sizes for round 7, since we found that many of our differences in coding decisions during the previous round were due having missed details instead of actively disagreeing about the meaning of a quote.

We calculated interrater reliability using Krippendorff's alpha, and our scores for each round are shown in Table 1. Since rounds 1 through 5 consisted of small sets of quotes, we calculated a reliability score for all coding decisions across each of these rounds prior to moving on to round 6. A "-" indicates that a theme was excluded from a round, while a "?" indicates that a theme was included in that round, but that no quotes were coded for it. A "?" is equivalent to an alpha score of 1, but we found that this distinction was useful to determine if we had actually encountered quotes for that theme during a particular set of rounds.

During each round, coding decisions were recorded in an online form. The form was split into multiple sections, with one question per theme. Table 2 outlines the structure of the survey and provides references to tables containing the codes for each theme. Some themes, such as "Uncertainty," are their own code and are can only be present or absent from a quote. Others, such as "Tools," can have multiple codes appear within a single quote. After each round, the first and third author met and refined the themes, making changes to the form as necessary. Each of these changes is reflected in the structure of Table 2 and the tables it references. The replication package contains an unabbreviated list of the themes and codes used during each round.

Table 1: Interrater reliability between the first and second author calculated using Krippendorff's alpha. The column "Survey Questions" includes references to sections or individual questions that have a direct correspondence to each theme, indicating that a theme is being used as a product of our qualitative analysis.

| Theme | Rounds 1-5 | Round 6 | Round 7 | Survey Questions |
|---|---|---|---|---|
| Stigma Against `unsafe` | **1** | 0.47 | ? | 3.26.0.2, 3.26.0.3 |
| Issues With Miri | 0.66 | 0.49 | **0.8** | 3.25.0.1 |
| Documented Invariants | 0.49 | **1** | 0 | 3.19.0.1 |
| Uncertainty | **0.86** | 0.66 | - | - |
| Interoperation | **0.83** | 0.68 | - | 3.21.0.1 |
| Types | 0.75 | **0.87** | - | 3.5.0.2 |
| Binding Generation Preferences | **0.74** | 0.35 | 0.5 | 3.21, 3.23 |
| Domain | **0.71** | 0.59 | 0.53 | 3.4.0.1, 3.4.0.2 |
| Found a bug? | 0.45 | 0.65 | **0.77** | 3.24.0.4, 3.25.0.3 |
| Why `unsafe`? | 0.33 | 0.23 | **0.7** | 3.11 |
| Tools | **0.73** | **0.74** | - | 3.24.0.1, 3.24.0.2 |
| Did they expose an `unsafe` API? | ? | 0 | 0 | 3.17.0.1 |
| `unsafe` Operations | 0.63 | 0.37 | 0.48 | 3.5.0.2 |
| Challenges with Bug-finding | ? | ? | - | - |
| `unsafe` Data Structure | 0.58 | -0.01 | - | - |

Table 2: Themes and theme titles used during inter-rater reliability. The "Type" column indicates the method used to indicate that a code for that theme was present. Themes of type "Yes / No" were recorded by indicating their presence or absence by selecting "Yes," "N/A", or leaving the prompt unselected. Themes that are "Select One" or "Select All" have multiple constituent codes, where either one or all codes that apply could be selected, respectively. For themes with multiple codes, the "Codes" column includes a reference to a table of codes for that theme located further in the Appendix.

| **Theme** | **Codes** | **Type** | **Rounds** | **Title** |
|---|---|---|---|---|
| Bug-finding | Table 3 | Select One | 1-6 | Challenges with bug-finding |
| Bugs | Table 4 | Select All | 7 | Did the participant encounter any of these bugs? |
| | | | 1-6 | Found a bug? |
| Documented Invariants | N/A | Yes / No | 2-7 | Did they document pre and post-conditions or invariants for safety of unsafe code? |
| | | | 1 | Documented pre and postconditions for safety? |
| Domain | Table 5 | Select One | 1-7 | Domain |
| Generation vs. Validation | Table 6 | Select One | 1-6 | Binding generation preferences |
| | | | 7 | Did the participant express a preference for a certain type of binding generation? |
| Interoperation | | Select All | 1-6 | Interoperation with: |
| Issues with Miri | Table 8 | Select All | 7 | Did the participant encounter any of these issues with Miri? |
| | | | 1-6 | Issues with Miri |
| Shifting Ground | N/A | Yes / No | 3-6 | Shifting ground - Did they express uncertainty about Rust's semantics or describe how they are either changing or unclear? |
| | | | 2 | Did they express uncertainty about Rust's semantics? |
| | | | 1 | Uncertainty about Rust's semantics |
| Stigma | N/A | Yes / No | 2-7 | Did they speak about a stigma against unsafe or express shame or negative feelings about unsafe code? |
| | | | 1 | Stigma Against Unsafe |
| Tools | Table 10 | Select All | 1-6 | Tools |
| Types | Table 11 | Select All | 1-6 | Types |
| Unsafe API | N/A | Yes / No | 2-7 | Did they expose an unsafe API? |
| | | | 1 | Exposed an unsafe API? |
| Unsafe Data Structure | N/A | Yes / No | 2-6 | Did they describe an unsafe data structure? |
| | | | 1 | Unsafe data structure? |
| Unsafe Operations | Table 12 | Select All | 1 | Low-level operations |
| | | | 2-7 | Did they use any of these unsafe operations? |
| Why unsafe? | Table 13 | Select All | 1-7 | Why unsafe? |

Table 3: Codes for the “Bug Finding” theme, which indicate challenges that participants encountered with bug-finding tools such as false positives or false negatives.

| **Bug Finding** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-6 | False Positive, N/A, X missed bug |

Table 4: Codes for the “Bugs” Theme, which indicate different types of bugs that were mentioned by interview participants. Cells spanning multiple rows in the “Rounds” column indicate that codes were either merged, split, or replaced in later rounds.

| **Bugs** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-5 | Mismatched Bindings |
| 6-7 | Mismatched FFI or Intrinsic Declaration |
| 1-7 | Data Races, Deallocate Foreign Memory in Rust, Double Free, Improper Alignment, Incorrect Linking, Incorrect Type Casting, Memory Leak, Undefined Behavior, Unwinding Across FFI Boundaries, Use After Free, Using Uninitialized Memory, Zero-Sized Allocation, From-Raw Wrong Pointer |
| 4-7 | Excessive Resource Use, Out of Bounds Access |
| 6-7 | Logical Error, Other, Stacked/Tree Borrows Violation |

Table 5: Codes for the “Domain” Theme, which indicates the different types of application domains mentioned by interview participants. Cells spanning multiple rows in the “Rounds” column indicate that codes were replaced or split in later rounds.

| **Domain** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1 | UEFI/Bootloader |
| 1-7 | Embedded Systems |
| 1-3 | Web development |
| 4-7 | Web Application Development, Web Browser Development |
| 1-7 | Databases, JIT Compilation, N/A, Operating Systems, Windows/WinAPI |
| 4-7 | Game Development |
| 6-7 | Other |

Table 6: Codes for the “Generation vs. Validation” Theme, which indicate interview participants’ preferences for using certain type of methods to create declarations for foreign functions.

| **Generation vs. Validation** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-5 | N/A |
| 1-6 | Edit the output of a binding generation tool, Use the results from a binding generation tool without modification, Write bindings manually |

Table 7: Codes for the “Interoperation” Theme, which indicate the languages that participants used alongside Rust in multi-language applications.

| **Interoperation** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-6 | ”WASM / JavaScript / TypeScript”, C, C++, Java, Python |

Table 8: Codes for the “Issues with Miri” theme, which indicate the difficulties that participants had when validating their test cases using Miri. Cells spanning multiple rows in the “Rounds” column indicate that codes were either merged, split, or replaced in later rounds.

| **Issues with Miri** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-3 | Lack of support for foreign function calls |
| 1-3 | Lack of support for inline assembly |
| 4-7 | Lack of support for a feature or construct |
| 1-7 | Tree Borrows/Stacked Borrows incorrectly considers my unsafe to be in violation of Rust’s aliasing model. |
| 1-7 | Runtime overhead |

Table 9: Codes for the “Low-Level Operations” theme, which indicate a subset of operations that Rust developers may use in embedded or OS development contexts. After the first round, this theme was merged with the theme “Unsafe Operations”, which is described in Table 12

| **Low-Level Operations** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1 | Custom allocator, Inline assembly, Instrinsics, Pointer arithmetic / integer to pointer conversion, System calls |

Table 10: Codes for the “Tools” theme, which indicate the different development tools mentioned by interview participants.

| **Tools** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-6 | ASAN, CXX, Clippy, Crucible, DHAT, Diplomat, Emscripten, Kani, Loom, MSAN, Miri, Prusti, PyO3, Shuttle, TSAN, UBSAN, UniFFI, Valgrind, bindgen, cargo fuzz, cbindgen, libfuzzer, wasm-bindgen |
| 2-6 | JNA |
| 6 | Other |

Table 11: Codes for the “Types” theme, which indicate the different Rust data types mentioned by interview participants. The code `ManuallyDrop<T>` was unintentionally left out of round 6.

| **Types** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-5 | `ManuallyDrop<T>` |
| 1-6 | `Box<T>`, `Cell<T>`, `ExactLen`, `MaybeUninit<T>`, `Mutex`, `NonNull<T>`, `OnceCell<T>`, `Pin<T>`, `RWLock<T>`, `Rc/Arc<T>`, `RefCell<T>`, `Send`, `Sync`, `TrustedLen`, `Vec<T>`, Weak Reference (`Weak<T>`) |
| 3-6 | Static Variables |
| 6 | `Layout`, `LazyCell<T>`, `Other`, `UnsafeCell<T>` |

Table 12: Codes for the “Unsafe Operations” theme, which indicate the different types of `unsafe` operations mentioned by interview participants. Cells spanning multiple rows in the “Rounds” column indicate that codes were either merged, split, or replaced in later rounds.

| **Unsafe Operations** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-5 | Pointer arithmetic / integer to pointer conversion |
| 1-2 | Custom allocator |
| 3-7 | Replaced Rust’s default allocator with a custom allocator |
| 1-7 | Inline assembly, Instrinsics, System calls |
| 3-7 | Raw allocation in Rust |
| 4-7 | Transmute |
| 6-7 | Multiple / cyclic aliasing, Other |

Table 13: Codes for the “Why `unsafe`?” theme, which indicate the different motivations that participants cited for their use of `unsafe` code. Cells spanning multiple rows in the “Rounds” column indicate that a code was replaced in later rounds.

| **Why unsafe?** | |
|---|---|
| **Rounds** | **Code(s)** |
| 1-2 | To circumvent the constraints imposed by the API of a dependency or 3rd party |
| 1-5 | To circumvent the constraints imposed by Rust’s type system |
| 1-5 | To interact with hardware the operating system or other low-level external interfaces |
| 1-5 | To interoperate with foreign code |
| 1-7 | For fun |
| 1-7 | To increase performance |
| 3-7 | Unsafe as documentation |
| 1-2 | Unsafe is more ergonomic |
| 3-5 | Unsafe is more ergonomic or easier to use |
| 6-7 | More ergonomic or easier to use |
| 6-7 | Impossible or no other choice |
| 6-7 | Other (short answer) |

## 3 Community Survey

### 3.1 Consent

*Prior to this section respondents were shown our consent script, which is available as part of our replication package.*

1. [ *required, yes/no* ] I am age 18 or older.
2. [ *required, yes/no* ] I have read and understand the information above.
3. [ *required, yes/no* ] I want to participate in this research and continue with the survey.

### 3.2 Eligibility

1. [ *required,* $\geq 1$ ] How many years of experience do you have using Rust?
   - See Table 14
2. [ *required, yes/no* ] Have you written, edited, read, audited, or engaged with unsafe Rust code in any way?

### 3.3 Background

1. [ *short answer* ] How many years have you engaged with programming or the field of software engineering at-large? This includes acting in a professional capacity and as a hobby.
   - See Table 14
2. [ *short answer* ] How many years of experience do you have using C?
   - See Table 14
3. [ *short answer* ] How many years of experience do you have using C++?
   - See Table 14
4. [ *select one* ] How frequently do you engage with unsafe Rust code in any way?
   - (69) - *Monthly*
   - (59) - *Weekly*
   - (35) - *Yearly*
   - (28) - *Daily*

Table 14: Survey respondents' years of experience in software engineering and with multiple programming languages. Values are grouped by where each respondent had first heard about the survey. Each respondent could select multiple recruitment sources. The "#" column indicates the number of participants that reached the survey from each source. The column "Software" corresponds to participants' answer to Question 3.3.0.1. Values are formatted as follows: "$min - max(mean \pm st.dev)$".

| **Recruitment** | # | **Software** | **C** | **C++** | **Rust** |
|---|---|---|---|---|---|
| Rust Forums | 22 | 3 - 42 (16.8 ± 10.3) | 0 - 20 (7.6 ± 6.6) | 0 - 20 (5.6 ± 5.7) | 1 - 9 (4.7 ± 2.3) |
| Reddit (/r/rust) | 101 | 2 - 43 (13.3 ± 8.5) | 0 - 30 (7.2 ± 6.9) | 0 - 30 (5.9 ± 6) | 1 - 9 (4.1 ± 2.1) |
| Rust Discord | 12 | 3 - 15 (8.9 ± 4.1) | 0 - 7 (2.8 ± 2.6) | 0 - 6 (3.6 ± 2.2) | 1 - 7 (3.3 ± 2) |
| "This Week in Rust" | 63 | 0 - 41 (15.5 ± 9.4) | 0 - 35 (6.3 ± 7.4) | 0 - 33 (5.2 ± 6.6) | 1 - 10 (4.1 ± 2.3) |
| Other | 9 | 3 - 41 (17 ± 10.9) | 0 - 33 (12 ± 10.3) | 1 - 19 (9.7 ± 6.9) | 1.5 - 5 (3.2 ± 1.2) |
| All (Survey) | 203 | 0 - 43 (14.3 ± 9.1) | 0 - 35 (6.9 ± 7.2) | 0 - 33 (5.7 ± 6.1) | 1 - 10 (4.1 ± 2.2) |

- `(12)` - *Less than once a year*

5. Have you ever committed unsafe Rust code to a public GitHub repository?
   - `(146)` - *Yes*
   - `( 57)` - *No*
6. Do you regularly write new Rust code or edit existing Rust code within an unsafe block or function?
   - `(111)` - *Yes*
   - `( 92)` - *No*
7. [ *select one* ] How frequently do you write new Rust code or edit existing Rust code within an unsafe block or function?
   - `(83)` - *Monthly*
   - `(42)` - *Yearly*
   - `(40)` - *Weekly*
   - `(22)` - *Less than once a year*
   - `(16)` - *Daily*

## 3.4 Domain

1. [ *select all* ] Which of the following types of applications have you developed or contributed to in any capacity using any programming language?
   - `(148)` - *Web applications - Backend*
   - `(144)` - *Data structures and language-level primitives*
   - `(129)` - *Serialization & deserialization*
   - `(115)` - *Networking & distributed systems*
   - `(113)` - *Web applications - Frontend*
   - `(112)` - *Embedded systems*
   - `(107)` - *Interpreters & runtime systems*
   - `(105)` - *Image & Data processing*
   - `(87)` - *Game development*
   - `(72)` - *Operating systems*
   - `(59)` - *Database systems*
   - `(58)` - *Non-JIT Compilation*
   - `(43)` - *Static analysis*
   - `(29)` - *Other*
   - `(15)` - *JIT Compilation*
   - `(14)` - *Manufacturing*
   - `( 9)` - *Web browsers*
2. [ *select all* ] Which of the following types of applications have you developed or contributed in any capacity using Rust?
   - `(109)` - *Data structures and language-level primitives*
   - `(105)` - *Web applications - Backend*
   - `(104)` - *Serialization & deserialization*
   - `( 77)` - *Networking & distributed systems*
   - `( 68)` - *Image & Data processing*
   - `( 67)` - *Embedded systems*
   - `( 65)` - *Interpreters & runtime systems*
   - `( 58)` - *Game development*
   - `(39)` - *Web applications - Frontend*
   - `(38)` - *Operating systems*
   - `(35)` - *Non-JIT Compilation*
   - `(33)` - *Database systems*
   - `(27)` - *Static analysis*
   - `(21)` - *Other*
   - `(10)` - *JIT Compilation*
   - `( 6)` - *Manufacturing*
   - `( 2)` - *Web browsers*
3. Have you rewritten part or all of an application in Rust that was originally written in another language?
   - `(147)` - *Yes*

- ( 56) - *No*

4. Have you authored or regularly contributed to a Rust crate that is published on crates.io?
   - (127) - *Yes*
   - ( 76) - *No*

5. Have you authored or regularly contributed to a Rust crate not published on crates.io?
   - (167) - *Yes*
   - ( 36) - *No*

6. [ *select one* ] When you first started using Rust, was it easy or difficult to write code that passed the borrow checker?
   - ( 8) - *Extremely difficult*
   - (90) - *Somewhat difficult*
   - (50) - *Neither easy nor difficult*
   - (45) - *Somewhat easy*
   - (10) - *Extremely easy*

## 3.5 Unsafe Features

1. [ *select all* ] Which of the following Rust types have you used in an unsafe context for any purpose?
   - (141) - `MaybeUninit`
   - (104) - `Box`
   - ( 95) - `ManuallyDrop`
   - ( 88) - `UnsafeCell`
   - ( 63) - `Pin`
   - ( 62) - `Arc`
   - ( 53) - `Mutex`
   - (43) - `Cell`
   - (38) - `OnceCell`
   - (37) - `Rc`
   - (31) - `RefCell`
   - (25) - `Other`
   - (18) - `LazyCell`

2. [ *select all* ] Which of the following unsafe Rust features have you used?
   - (179) - *Calling unsafe functions written in Rust*
   - (166) - *Dereferencing a raw pointer*
   - (148) - *mem::transmute*
   - (143) - *Calling foreign functions*
   - (118) - *Pointer arithmetic*
   - (111) - *Implementing an unsafe trait*
   - ( 89) - *Mutable static variables*
   - ( 83) - *Manual memory allocation*
   - ( 77) - *In-place initialization or updates*
   - (77) - *System calls*
   - (69) - *Declaring an unsafe trait*
   - (67) - *Accessing the fields of a union type*
   - (67) - *Inline assembly*
   - (61) - *Implementing a custom allocator*
   - (58) - *Intrinsics*
   - (52) - *Self-referential structs*
   - (31) - *Global assembly*
   - (19) - *Cyclic aliasing patterns*
   - ( 9) - *Other*

3. Have you contributed to a Rust crate that is designed to be used from at least one other language?
   - ( 90) - *Yes*
   - (113) - *No*

## 3.6 UnsafeCell

Questions in this section were only shown to respondents if they had selected "`UnsafeCell`" in their response to Question 3.5.0.1 of the survey.

1. [ *select one* ] When you choose to use `UnsafeCell`, how often are you certain that it is necessary to avoid undefined behavior?
   - `( 1) -` *Never*
   - `( 8) -` *Sometimes*
   - `( 6) -` *About half the time*
   - `(39) -` *Most of the time*
   - `(34) -` *Always*

## 3.7 Calling Unsafe Functions

*This section was only shown to respondents if they had selected "Calling unsafe functions written in Rust" or "Calling foreign functions" in their response to Question 3.5.0.1 of the survey.*

1. [ *select one* ] When you use an `unsafe` API, how often do you look for documentation to ensure that you meet all of its requirements for safety and correctness?
   - `(  3) -` *Never*
   - `( 12) -` *Sometimes*
   - `( 13) -` *About half the time*
   - `( 48) -` *Most of the time*
   - `(117) -` *Always*
2. [ *select one* ] When you use an `unsafe` API, how often do you insert runtime checks to ensure that you meet its requirements for safety and correctness?
   - `(19) -` *Never*
   - `(64) -` *Sometimes*
   - `(29) -` *About half the time*
   - `(56) -` *Most of the time*
   - `(25) -` *Always*

## 3.8 Intrinsics

*This section was only shown to respondents if they had selected "Intrinsics" in their response to Question 3.5.0.1 of the survey.*

1. [ *select all* ] Which of the following types of intrinsics have you used in Rust applications?
   - `(47) -` *Architecture-specific or SIMD intrinsics*
   - `(26) -` *Atomics*
   - `(17) -` *Const intrinsics*
   - `(12) -` *Volatiles*
   - `( 7) -` *Other*
2. Do you use runtime checks in Rust applications, such as the `is_x86_feature_detected!` macro, to determine when certain hardware-specific features are supported?
   - `(22) -` *Yes*
   - `(36) -` *No*

## 3.9 ManuallyDrop + MaybeUninit

*This section was only shown to respondents if they had selected "`MaybeUninit`" and "`ManuallyDrop`" in their response to Question 3.5.0.2 of the survey.*

1. [ *select one* ] In Rust, how often is your use of `ManuallyDrop` associated with your use of `MaybeUninit`?
   - `(21) -` *Never*
   - `(37) -` *Sometimes*
   - `(12) -` *Unsure*
   - `( 3) -` *About half the time*
   - `(14) -` *Most of the time*
   - `( 1) -` *Always*

## 3.10 Box

*This section was only shown to respondents if they had selected "`Box`" in their response to Question 3.5.0.2 of the survey.*

1. According to Rust's current semantics, if you re-wrap a raw pointer as a `Box` using `Box::from_raw`, and then you move the `Box`, the raw pointer becomes invalid. Has this caused problems or architectural challenges in the applications that you have contributed to?
   - `(18) -` *Yes*
   - `(86) -` *No*
2. Have you ever encountered memory leaks due to leaking a raw pointer using `Box::into_raw()` and then forgetting to re-wrap it using `Box::from_raw()`?
   - `(24) -` *Yes*
   - `(80) -` *No*

## 3.11 Why unsafe?

1. [ *select all* ] Think about the situations where you have typically used `unsafe`. Which of the following reasons have motivated you to do so?
   - `(159) -` *I am not aware of a safe alternative at any level of ease-of-use or performance.*
   - `( 99) -` *I could use a safe pattern but unsafe is faster or more space-efficient.*
   - `( 38) -` *I could use a safe pattern but unsafe is easier to implement or more ergonomic.*
   - `( 22) -` *Other*

## 3.12 Why unsafe? - No clear alternative

*This section was only shown to respondents if they had selected "I am not aware of a safe alternative at any level of ease-of-use or performance" in their response to Question 3.11.0.1 of the survey.*

1. [ *select all* ] Which of the following `unsafe` features have you used because you were not aware of a safe alternative?
   - `(99) -` *Calling foreign functions*
   - `(99) -` *Calling `unsafe` functions written in Rust*
   - `(88) -` *Dereferencing a raw pointer*
   - `(76) -` `mem::transmute`
   - `(66) -` *Implementing an `unsafe` trait*
   - `(47) -` *Pointer arithmetic*
   - `(41) -` *Inline assembly*

- (41) - *System calls*
- (33) - *Declaring an `unsafe` trait*
- (32) - *Accessing the fields of a union type*
- (32) - *Intrinsics*
- (29) - *Implementing a custom allocator*
- (29) - *Manual memory allocation*
- (28) - *In-place initialization or updates*
- (27) - *Mutable static variables*
- (22) - *Self-referential structs*
- (14) - *Global assembly*
- ( 7) - *Cyclic aliasing patterns*
- ( 2) - *Other*

2. [ *select one* ] When you feel that you have no clear alternative other than using `unsafe`, how often are you certain that it would be completely impossible to accomplish this task using a safe design pattern?

   - ( 3) - *Never*
   - (24) - *Sometimes*
   - ( 6) - *Unsure*
   - (25) - *About half the time*
   - (79) - *Most of the time*
   - (22) - *Always*

*The next question was only shown to respondents if they had selected any of the following types in their response to Question 3.5.0.2 of the survey.*

3. [ *select all* ] Which of the following Rust types have you used because you were not aware of a safe alternative?

   - (75) - *MaybeUninit*
   - (48) - *UnsafeCell*
   - (46) - *ManuallyDrop*
   - ( 5) - *Other*

## 3.13 Why unsafe? - Ergonomics

*This section was only shown to respondents if they had selected "I could use a safe pattern but unsafe is easier to implement or more ergonomic" in their response to Question 3.11.0.1 of the survey.*

1. [ *select all* ] Which of the following `unsafe` features have you used because it was easier to implement or more ergonomic than a safe alternative?

   - (25) - *mem::transmute*
   - (18) - *Dereferencing a raw pointer*
   - (17) - *Calling unsafe functions written in Rust*
   - (14) - *Calling foreign functions*
   - (13) - *Mutable static variables*
   - (10) - *Pointer arithmetic*
   - ( 9) - *In-place initialization or updates*
   - ( 8) - *Implementing an unsafe trait*
   - ( 6) - *Manual memory allocation*
   - (6) - *Self-referential structs*
   - (5) - *Declaring an unsafe trait*
   - (5) - *Implementing a custom allocator*
   - (4) - *Accessing the fields of a union type*
   - (3) - *Cyclic aliasing patterns*
   - (3) - *Intrinsics*
   - (3) - *System calls*
   - (2) - *Global assembly*
   - (2) - *Inline assembly*

*This question was only shown to respondents if they had selected any of the following types in their response to Question 3.5.0.2 of the survey.*

2. [ *select all* ] Which of the following Rust types have you used because it was easier to implement or more ergonomic than a safe alternative?

- (15) - `MaybeUninit`
- (13) - `ManuallyDrop`
- (10) - `UnsafeCell`
- ( 1) - *Other*

## 3.14 Why unsafe? - Performance

*This section was only shown to respondents if they had selected "I could use a safe pattern but unsafe is easier to implement or more ergonomic" in their response to Question 3.11.0.1 of the survey.*

1. [ *select all* ] Which of the following unsafe features have you used because it performed faster or was more space-efficient than a safe alternative?
   - (55) - `mem::transmute`
   - (52) - *Calling unsafe functions written in Rust*
   - (51) - *Dereferencing a raw pointer*
   - (41) - *In-place initialization or updates*
   - (34) - *Pointer arithmetic*
   - (29) - *Intrinsics*
   - (22) - *Manual memory allocation*
   - (21) - *Implementing an* `unsafe` *trait*
   - (21) - *Inline assembly*
   - (20) - *Self-referential structs*
   - (19) - *Calling foreign functions*
   - (18) - *Accessing the fields of a union type*
   - (18) - *Mutable static variables*
   - (13) - *Implementing a custom allocator*
   - (12) - *Declaring an* `unsafe` *trait*
   - (11) - *System calls*
   - (10) - *Cyclic aliasing patterns*
   - ( 5) - *Global assembly*
   - ( 2) - *Other*

*The next question was only shown to respondents if they had selected any of the following types in their response to Question 3.5.0.2 of the survey.*

2. [ *select all* ] Which of the following Rust types have you used because it performed faster or was more space efficient than a safe alternative?
   - (60) - `MaybeUninit`
   - (29) - `UnsafeCell`
   - (25) - `ManuallyDrop`
   - ( 2) - *Other*

3. [ *select one* ] When you choose to use unsafe because it performs faster or is more space efficient, how often do you measure the difference?
   - (13) - *Never*
   - (22) - *Sometimes*
   - (18) - *About half the time*
   - (20) - *Most of the time*
   - (26) - *Always*

4. [ *select one* ] When you use unsafe because it performs faster or is more space efficient, are you typically introducing small-scale optimizations into existing applications, or are you designing large-scale abstractions that are purpose-built for performance?
   - (42) - *Both*
   - (24) - *Small-scale optimizations*
   - (20) - *Large-scale components purpose-built for performance.*
   - ( 8) - *Unsure*
   - ( 5) - *Other*

## 3.15 Performance VS Safety

1. [ *select one* ] You are writing an arbitrary Rust application, and you have the opportunity to increase its speed or space-efficiency using unsafe code. You are certain that the unsafe code is sound, and your test cases pass Miri, but the performance increase is marginal. Which would you choose:
    - `(137)` - *Use safe*
    - `( 38)` - *Unsure*
    - `( 28)` - *Use unsafe*

*The next question was only shown to participants if they had answered "Yes" to either Question 3.4.0.3 or Question 3.4.0.4.*

2. [ *select one* ] You are writing a Rust crate and you have the opportunity to increase its speed or space-efficiency using unsafe code. You are certain that the unsafe code is sound, and your test cases pass Miri, but the performance increase is marginal. Which would you choose:
    - `(130)` - *Use safe*
    - `( 31)` - *Unsure*
    - `( 23)` - *Use unsafe*

## 3.16 Local Minimal Unsafe

1. [ *select one* ] If another Rust developer at your skill level selected a random `unsafe` block or function from code that you have written, would it be easy or difficult for them to understand?
    - `(  4)` - *Extremely difficult*
    - `( 27)` - *Somewhat difficult*
    - `( 52)` - *Neither easy nor difficult*
    - `(101)` - *Somewhat easy*
    - `( 19)` - *Extremely easy*
2. [ *select all* ] Select each of the following features or attributes that you have chosen to enable in a Rust module.
    - `(68)` - `unsafe_block_in_unsafe_fn`
    - `(47)` - `deny(unsafe_code)`
    - `(47)` - `forbid(unsafe_code)`
    - `(13)` - `allow(improper_ctypes)`
    - `( 7)` - `allow(improper_ctypes_definitions)`
    - `( 4)` - `const_raw_ptr_deref`
3. [ *select one* ] How often do you refactor Rust applications to remove unsafe code?
    - `( 52)` - *Never*
    - `(126)` - *Sometimes*
    - `( 12)` - *About half the time*
    - `( 12)` - *Most of the time*
    - `(  1)` - *Always*
4. [ *select one* ] How often do you choose to avoid using an unsafe API when a safe alternative exists?
    - `(  5)` - *Never*
    - `( 16)` - *Sometimes*
    - `( 14)` - *About half the time*
    - `(121)` - *Most of the time*
    - `( 47)` - *Always*

### 3.17 Encapsulation

1. Have you exposed an unsafe API to other Rust developers?
    - `( 92) -` *Yes*
    - `(111) -` *No*
2. Have you exposed a safe API to other Rust developers that encapsulates unsafe code?
    - `(180) -` *Yes*
    - `( 23) -` *No*

### 3.18 Encapsulation - Unsafe API

*This section was only shown to respondents if they had selected "Yes" in their response to Question 3.17.0.1.*

1. [ *select all* ] Which of the following reasons have motivated you to expose an unsafe API to users?
    - `(69) -` *Impossible to encapsulate without imposing safety requirements on the user*
    - `(51) -` *To provide a more performant equivalent of a safe API*
    - `(12) -` *To provide a more ergonomic equivalent of a safe API*
    - `( 5) -` *Other*
2. [ *select one* ] When you expose an unsafe API, how often are its users responsible for meeting certain requirements for correctness and safety?
    - `( 1) -` *Never*
    - `( 3) -` *Sometimes*
    - `( 3) -` *Unsure*
    - `( 4) -` *About half the time*
    - `(23) -` *Most of the time*
    - `(58) -` *Always*

### 3.19 Encapsulation - Unsafe API - Invariants

*Questions in this section were only shown to respondents if they had not selected "Never" in their response to Question 3.18.0.2.*

1. [ *select one* ] When you expose an unsafe API to users, and it is their responsibility to ensure that certain requirements are met, how often do you document these requirements?
    - `(68) -` *Always*
    - `(14) -` *Most of the time*
    - `( 6) -` *About half the time*
    - `( 3) -` *Sometimes*
2. [ *select one* ] When you expose an unsafe API to other users, and it is their responsibility to ensure that certain requirements are met, how often do you include runtime checks for these requirements?
    - `(17) -` *Never*
    - `(50) -` *Sometimes*
    - `(11) -` *About half the time*
    - `(10) -` *Most of the time*
    - `( 3) -` *Always*

## 3.20 Encapsulation - Safe API

*This section was only shown to respondents if they had selected "Yes" in their response to Question 3.17.0.2.*

1. [ *select one* ] When you expose a safe API for unsafe code, how often do you include runtime checks to ensure that its requirements for correctness and safety are met?
   - (15) - *Never*
   - (40) - *Sometimes*
   - (23) - *About half the time*
   - (56) - *Most of the time*
   - (46) - *Always*
2. [ *select one* ] When you expose a safe API for unsafe code, how often are all of its requirements for correctness and safety satisfied by the properties of Rust's type system?
   - ( 3) - *Never*
   - (28) - *Sometimes*
   - (15) - *Unsure*
   - (25) - *About half the time*
   - (68) - *Most of the time*
   - (41) - *Always*

## 3.21 FFI - Background

*This section was only shown to respondents if they had selected "Calling foreign functions" in their response to Question 3.5.0.2.*

1. [ *select all* ] When you call foreign functions from Rust or write Rust bindings to foreign functions, which languages are these functions written in?
   - (131) - *C*
   - ( 82) - *C++*
   - ( 32) - *Assembly*
   - ( 26) - *Python*
   - ( 18) - *JavaScript*
   - ( 16) - *Other*
   - ( 10) - *Java*
   - ( 10) - *TypeScript*
   - (6) - *Swift*
   - (4) - *C#*
   - (4) - *Go*
   - (3) - *Dart*
   - (1) - *Haskell*
   - (1) - *OCaml*
   - (1) - *Ruby*
2. [ *select one* ] Do you write or generate bindings to foreign functions?
   - (67) - *I write bindings manually and generate bindings using a tool*
   - (42) - *I generate bindings using a tool*
   - (21) - *I write bindings manually*
   - (13) - *I do not write or generate bindings*
3. [ *select one* ] Do you trust FFI bindings that are written by hand?
   - ( 4) - *Definitely not*
   - ( 8) - *Probably not*
   - (71) - *Might or might not*

- `(48)` - *Probably yes*
- `(12)` - *Definitely yes*

4. [ *select one* ] Do you trust FFI bindings that are generated by a tool?
    - `(89)` - *Probably yes*
    - `(28)` - *Might or might not*
    - `(23)` - *Definitely yes*
    - `( 3)` - *Probably not*
5. [ *select one* ] Have you ever encountered incorrect FFI bindings in a Rust application?
    - `(63)` - *No*
    - `(49)` - *Yes*
    - `(31)` - *Unsure*

*This question was only shown to respondents if they had selected "Yes" to the previous question.*

6. [ *select one* ] Have these incorrect bindings typically been written manually, generated by a tool, or both?
    - `(18)` - *Both*
    - `(14)` - *Written manually*
    - `( 9)` - *Generated by a tool*
    - `( 8)` - *Unsure*

## 3.22 FFI - Memory Model

*This section was only shown to respondents if they had selected "Calling foreign functions" in their response to Question 3.5.0.2.*

*The next question was only shown to respondents if they had selected "`Box`", "`Arc`", or "`Rc`" in their response to Question 3.5.0.1.*

1. [ *select all* ] Select each of the following memory container types that you have used to allocate memory for a foreign function call. This process includes creating the container, exposing it as a raw pointer using `into_raw()`, and then passing it into a foreign function.
    - `(78)` - `Box`
    - `(24)` - `Arc`
    - `(11)` - *Other*
    - `( 7)` - `Rc`
2. [ *select one* ] When you receive a raw pointer to memory allocated by an FFI call, how often do you store it as an `UnsafeCell`?
    - `(92)` - *Never*
    - `(35)` - *Sometimes*
    - `(11)` - *Most of the time*
    - `( 5)` - *About half the time*
3. [ *select one* ] Do you pass Rust's abstract data types (structs, enums) by value across FFI boundaries?
    - `(79)` - *No*

- `(44)` - *Yes*
- `(20)` - *Unsure*

4. [ *select one* ] Do you convert raw pointers to memory allocated by FFI calls into safe references, such as `&T` or `&mut T`?
   - `(86)` - *Yes*
   - `(35)` - *No*
   - `(22)` - *Unsure*
5. [ *select one* ] How often do you intentionally avoid passing Rust's abstract data types (structs, enums) by value across FFI boundaries?
   - `( 6)` - *Never*
   - `(30)` - *Sometimes*
   - `(28)` - *Unsure*
   - `( 9)` - *About half the time*
   - `(38)` - *Most of the time*
   - `(32)` - *Always*
6. [ *select one* ] How often do you intentionally avoid converting raw pointers to memory allocated by FFI calls into safe references, such as `&T` or `&mut T`?
   - `(14)` - *Never*
   - `(49)` - *Sometimes*
   - `(31)` - *Unsure*
   - `(14)` - *About half the time*
   - `(21)` - *Most of the time*
   - `(14)` - *Always*

## 3.23 FFI - Bindings - Automation

*This section was only shown to respondents if they had selected "I generate bindings using a tool" or "I write bindings manually and generate bindings using a tool" in their response to Question 3.21.0.2.*

1. [ *select all* ] Which of the following binding generation tools have you used?
   - `(86)` - *bindgen*
   - `(52)` - *wasm-bindgen*
   - `(47)` - *cbindgen*
   - `(42)` - *PyO3*
   - `(35)` - *CXX*
   - `(13)` - *Emscripten*
   - `(12)` - *Other*
   - `( 2)` - *Diplomat*
   - `( 1)` - *UniFFI*

*The next question was only shown to participants if they had selected "bindgen" in their response to Question 3.23.0.1*

2. [ *select one* ] Do you typically generate bindings locally and check them in, or do you generate them as a CI step?
   - `(41)` - *Locally*
   - `(13)` - *In CI*
   - `(13)` - *Other*

*The next question was only shown to participants if they had selected "wasm-bindgen" in their response to Question 3.23.0.1*

3. [ *select one* ] Has it typically been difficult or easy to configure wasm-bindgen?

   - `( 2)` - *Extremely difficult*
   - `(11)` - *Somewhat difficult*
   - `(15)` - *Neither easy nor difficult*
   - `(19)` - *Somewhat easy*
   - `( 5)` - *Extremely easy*

## 3.24 Validation

1. [ *select all* ] Which of the following formal methods tools have you used with Rust?

   - `(9)` - *Kani*
   - `(3)` - *Creusot*
   - `(3)` - *Prusti*
   - `(2)` - *Flux*
   - `(2)` - *Other*

2. [ *select all* ] Which of the following sanitizers, fuzzers, and dynamic analysis tools have you used for Rust applications?

   - `(127)` - *Miri*
   - `( 91)` - *Valgrind*
   - `( 53)` - *cargo fuzz*
   - `( 51)` - *AddressSanitizer (ASAN)*
   - `( 27)` - *ThreadSanitizer (TSAN)*
   - `( 26)` - *UndefinedBehaviorSanitizer (UBSAN)*
   - `( 23)` - *libFuzzer*
   - `( 17)` - *LeakSanitizer (LSAN)*
   - `(17)` - *MemorySanitizer (MSAN)*
   - `(15)` - *Loom*
   - `( 5)` - *Other*
   - `(3)` - *HWAddressSanitizer (HWASAN)*
   - `( 2)` - *KernelControlFlowIntegrity*
   - `( 2)` - *Shuttle*
   - `( 1)` - *ShadowCallStack*

3. [ *select one* ] How frequently do you use a debugger with a Rust application?

   - `(19)` - *Daily*
   - `(37)` - *Weekly*
   - `(58)` - *Monthly*
   - `(31)` - *Yearly*
   - `(32)` - *Less than once a year*
   - `(26)` - *Never*

4. [ *select all* ] Which of the following types of bugs and undefined behaviors have you encountered in a Rust application and were caused by unsafe code?

   - `(86)` - *Out-of-bounds access*
   - `(76)` - *Using uninitialized memory*
   - `(70)` - *Memory leak*
   - `(65)` - *Use after free*
   - `(64)` - *Null pointer dereference*
   - `(56)` - *Unaligned memory access*
   - `(55)` - *Violation of stacked or tree borrows*
   - `(44)` - *Double free*
   - `(43)` - *Incorrectly declared bindings to intrinsics or foreign functions.*
   - `(31)` - *External runtime or hardware interface does not behave according to its written specification.*
   - `(25)` - *Failure to meet an application's functional requirements.*

- `(25)` - *Unwinding across foreign stack frames without using an -unwind ABI*
- `(22)` - *Deallocating memory with an incorrect Layout*
- `(22)` - *Integer overflow*
- `(11)` - *Creating a zero-size allocation with an allocator that requires nonzero size*
- `(11)` - *Other*

5. [ *select one* ] How often do you write tests for Rust applications that use unsafe?
   - `(15)` - *Never*
   - `(47)` - *Sometimes*
   - `(18)` - *About half the time*
   - `(56)` - *Most of the time*
   - `(67)` - *Always*
6. [ *select one* ] How often do you audit your dependencies' use of unsafe?
   - `(96)` - *Never*
   - `(82)` - *Sometimes*
   - `(11)` - *About half the time*
   - `( 9)` - *Most of the time*
   - `( 5)` - *Always*
7. [ *select one* ] Select each of the following tools that you use to audit your dependencies.
   - `(62)` - *cargo-audit*
   - `(40)` - *cargo-update*
   - `(34)` - *cargo-deny*
   - `(22)` - *cargo-geiger*
   - `( 7)` - *cargo-vet*
8. [ *select one* ] One of your options for a dependency has an unsafe API. Are you likely or unlikely to avoid choosing this dependency if other options have a safe API?
   - `(13)` - *Extremely unlikely*
   - `(24)` - *Somewhat unlikely*
   - `(62)` - *Neither likely nor unlikely*
   - `(67)` - *Somewhat likely*
   - `(37)` - *Extremely likely*

## 3.25 Validation - Miri

*Questions in this section were only shown to participants if they had selected "Miri" in their response to Question 3.24.0.2.*

1. [ *select all* ] Which of the following issues have deterred you from using Miri to test a Rust application?
   - `(58)` - *Lack of support for foreign function calls*
   - `(35)` - *Slow performance*
   - `(24)` - *Lack of support for inline assembly*
   - `(13)` - *Other*
   - `( 5)` - *My unsafe code is in violation of Tree Borrows*

- `( 4)` - *My unsafe code is in violation of Stacked Borrows*

2. [ *select one* ] How often do you run test cases in Miri?

   - `(16)` - *Never*
   - `(57)` - *Sometimes*
   - `( 9)` - *About half the time*
   - `(28)` - *Most of the time*
   - `(17)` - *Always*

3. [ *select all* ] Which of the following bugs has Miri detected in applications that you have contributed to?

   - `(54)` - *Stacked Borrows Violation*
   - `(47)` - *Using uninitialized memory*
   - `(33)` - *Out-of-bounds access*
   - `(33)` - *Use-after-free*
   - `(30)` - *Unaligned memory access*
   - `(25)` - *Memory leak*
   - `(20)` - *Tree Borrows Violation*
   - `(19)` - *Data race*
   - `(19)` - *Null pointer dereference*
   - `( 7)` - *Other*

*The next question was only shown to participants if they had selected either "Stacked Borrows Violation" or "Tree Borrows Violation" in their answer to Question 3.25.0.3.*

4. [ *select one* ] When Miri detects a borrowing violation in your code, has it usually been easy or difficult to fix?

   - `( 3)` - *Extremely difficult*
   - `(20)` - *Somewhat difficult*
   - `(14)` - *Neither easy nor difficult*
   - `(16)` - *Somewhat easy*
   - `( 6)` - *Extremely easy*

## 3.26 Community and Culture

1. [ *select all* ] Select each of the following online communities where you have posted asking for advice related to either undefined behavior or the correct use of unsafe code.

   - `(44)` - *The Rust Community Discord*
   - `(41)` - *The Rust Programming Language Forums*
   - `(40)` - *GitHub*
   - `(37)` - *Reddit (/r/rust)*
   - `(14)` - *Other*

2. [ *select one* ] How do you perceive that the Rust community views unsafe code in general?

   - `( 13)` - *Extremely negatively*
   - `(112)` - *Somewhat negatively*
   - `( 59)` - *Neither positively nor negatively*
   - `( 10)` - *Unsure*
   - `(  6)` - *Somewhat positively*
   - `(  3)` - *Extremely positively*

3. [ *select one* ] How do you perceive that the Rust community views the type of unsafe code that you write?

   - `( 5)` - *Extremely negatively*
   - `(31)` - *Somewhat negatively*
   - `(58)` - *Neither positively nor negatively*
   - `(79)` - *Unsure*
   - `(27)` - *Somewhat positively*
   - `( 3)` - *Extremely positively*

4. [ *select one* ] How often is the Rust community's guidance and documentation adequate for you to know how to use unsafe correctly?

   - `(  2)` - *Never*
   - `( 31)` - *Sometimes*
   - `( 17)` - *Unsure*
   - `( 13)` - *About half the time*
   - `(128)` - *Most of the time*
   - `( 12)` - *Always*

5. [ *select one* ] Do you agree or disagree with the following statement:"No documentation is better than incorrect documentation."

   - `( 9)` - *Strongly disagree*
   - `(23)` - *Somewhat disagree*
   - `(31)` - *Neither agree nor disagree*
   - `(84)` - *Somewhat agree*
   - `(56)` - *Strongly agree*

## 3.27 Demographics

1. [ *select one* ] What is your age?

   - `(115)` - *18-29*
   - `( 57)` - *30-39*
   - `( 17)` - *40-49*
   - `(  7)` - *50-59*
   - `(  6)` - *Prefer not to answer*
   - `(  1)` - *60-69*

2. [ *select one* ] What is your highest completed level of education?

   - `(10)` - *Some high school*
   - `(19)` - *High school diploma/GED*
   - `(30)` - *Some college*
   - `(77)` - *Bachelor's degree*
   - `(48)` - *Master's degree*
   - `(15)` - *PhD*
   - `( 4)` - *Prefer not to answer*

3. [ *select all* ] What is your gender?

   - `(164)` - *Man*
   - `( 19)` - *Prefer not to disclose*

- `( 15) -` *Non-binary*
- `(  5) -` *Woman*

4. [ *select one* ] Select each of the affiliations that best describes your role as a Rust developer.
   - `( 35) -` *Academia*
   - `(124) -` *Industry*
   - `(146) -` *Open-source contributor*
   - `( 10) -` *Rust project member*
   - `( 10) -` *Other*
5. [ *select all* ] Where did you first hear about this survey?
   - `(101) -` *Reddit (/r/rust)*
   - `( 63) -` *"This Week in Rust"*
   - `( 22) -` *The Rust Programming Language Forums*
   - `( 12) -` *The Rust Programming Language Community Discord*
   - `(  9) -` *Other*

## 3.28 Personal Information - Consent

1. [ *yes/no* ] Do you wish to be entered into a drawing for a $250 gift card to Amazon, Starbucks, or Target? To be eligible, you will need to provide a valid username from GitHub or the Rust Programming Language Forums and an email address. You account must have activity prior to the start date of the survey.
2. [ *yes/no* ] Are you open to being contacted for follow-up surveys or interviews?
3. [ *yes/no* ] Did you participate in our initial interview study this spring?

## 3.29 Personal Information - Connection

*Questions in this section were only shown to respondents who had selected "Yes" in their response to Question 3.28.0.3.*

1. [ *yes/no* ] Would you be open to providing your name and email address so that we can associate your responses to this survey with your data from the interview study? By selecting 'Yes,' you consent to be contacted for clarification if the name and email address you provided do not match earlier data.

## 3.30 Personal Information - Drawing

*Questions in this section were only shown to participants if they had selected "Yes" in their response to Question 3.28.0.1.*

The online profile that you provide will only be used to prevent multiple survey responses from being submitted by the same user. It will not be published.

1. [ *short answer* ] Copy and paste a link to your profile page on either GitHub or the Rust Programming Language Forums.
2. [ *select one* ] Which type of gift card would you prefer?
   - *Amazon*
   - *Starbucks*
   - *Target*

### 3.31 Personal Information - Email

*Questions in this section were only shown to participants if they had selected "Yes" in their response to Question 3.28.0.1, Question 3.28.0.3, or Question 3.29.0.1.*

Your name and email are collected for the purpose of contacting you if you are eligible for compensation and are selected, if you consented to be contacted for follow-up surveys or interviews, or if you consented to associate your survey response with prior interview study data and the name and email you provide do not match what you previously provided. These identifiers will not be used for any other purposes and will not be published.

1. [ *short answer* ] What is your name?
2. [ *short answer* ] What is your email?