Interviewer
What have been your motivations for learning Rust and choosing to use it?

Participant
My motivations have been that as mostly a C++ developer throughout my career, the language had started accumulating too much cruft in a way that it was too difficult to get rid of and without having many initiatives to clean it up. In other words, I ended up in certain projects just finding myself more comfortable just doing C rather than C++ due to the ever-evolving nature of C++ and what seemed to me to me a not particularly useful direction. So for cases where the code was relatively low level and simple enough, I'd switch to C and C++ whenever the code base was bigger. But yeah, change from one edition to the next and rewrite good portions of your code base or it'll have a stink factor, which has been fortunate. And all the while, I didn't really feel that the language was moving in a direction that I felt particularly useful for the type of work that I was doing.

5:1 My motivations have...

Cruft

Interviewer
Gotcha. So in terms of, you mentioned the sort of stink, how did you So in terms of, you mentioned the sort of stink that you perceive with code if you aren't keeping it up to date to the latest version. Is that more of a code style thing or is it that there are breaking changes that occur once you're upgrading?

Participant
No, no, no, no. The C++ is maintained in a way that is very good at not breaking backwards compatibility. It's simply that the well-known, I forgot who said it, was it [name]? [name], I think, maybe. But within, or maybe it might have been [name], that within C++ lies a smaller, neater language trying to get out. And to an extent, all the developers that I've worked with who work with C++ tend to pick their own favorite sub-language within the language. And if you start working on large enough code bases, the intersection of the sub-languages means the whole language. And that makes for a relatively unruly and unmaintainable code base. So the stink comes primarily from that style, yes. But when the language is so big, you find out that style matters because it changes your ability to review code. And yeah, I mean, in extreme cases you have fully deprecated features, but like auto pointers and things like that, which I think by now are long gone in most code bases. But yeah, when I started coding C++, it looked very, very different. And yeah, function signatures look different nowadays. The use of the auto keyword is everywhere now. And should you initialize with brackets or braces? Well, that sort of depends on the style that you're after. When you start doing templates, you end up with lots of expansion. Rust is a way to get things to be cleaner. There's also another fundamental reason. And that is that I've had the requirements in a number of projects, but mostly because it's not uncommon for me to write libraries for other people. And libraries for other people are going to be used in environments that you have no control over. So having a

5:2 But within, or...

C++ is Difficult

⋮

Rust is ergonomic

dependency on the C++ runtime is often a hindrance to the.   distribution of the binaries. So it's happened in the past where I would have liked to write something in C++, but just tagging the standard library on top of it would have been difficult. And if you compile C++ without the standard library, you're basically down with C with templates and classes. You're basically doing C again, so you might as well do C. So Rust offers a way out of that, where you can statically link pretty much everything you need and go from there.

5:3 And if yo...

Feature Disparity

Interviewer
Gotcha. So in terms of memory safety, then, and the different memory models of Rust and C++, how was that transition for you as a C++ developer moving into a language where you have the borrow checker, where you have these extensive restrictions on aliasing and mutability?

Participant
7 I think these are ideas that ... Put it this way, C++ introduced a big, big change when it introduced move semantics. Move semantics make a lot of sense in a language that has resource acquisition as initialization. Rust sort of made it the default. It's a very unusual choice to make a default, but once you do, however, it makes a lot of other things really, really easy. So I think it was the right thing. So that initially made a whole bunch of stuff simpler, because now you stop having to worry about these double ampersands everywhere and copy constructors. Many of these things become easier, so it lowers the degree of complexity there. But then you find that as a C++ developer moving to Rust, there's many tricks that you'd like to do that you just don't get to do out of the box. Some of them are genuinely hard to pull off. I mean, things like referring to having a pointer into your own object becomes very, very hard, because of this assumption that everything is going to be movable by default. So there's still cases that you sort of know better and you go, okay, well, okay, how do I break out of the constraints imposed by the language? And the initial beginner response to that is just to breach out for unsafe, which is not wrong as a learning experience, I think, because it allows you to find where the edges of the language are. Once you start writing more production code, you realize that actually there's Cell and RefCell, and there's Pin, and there's all sorts of little utilities that allow you to, not necessarily always, but quite often address that problem.

5:5 But then yo...

Feature Disparity

5:6 So there's still cases...

Cell<T>

Easier or More Erg

No Other Choice

Pin<T>

RefCell<T>

12

13 **Interviewer**
Yeah.

14

15 **Participant**
And it's still not perfect. I mean, it's kind of trivially easy to write a smart pointer in C++, but it's not trivially easy to do it in Rust, for example. I think there's a couple of things that are, I don't know the details about it and I haven't really looked at it because I haven't, but if I wanted to write truly highly performant code, and that means that I really, really need to sweat memory allocation, then that is still an area where I feel that it's still harder to do in Rust than it would be in C++.

5:7 And it's still no...

C++ is Easier to Use

Interviewer
Gotcha. So the next question is more specific to unsafe. So what have been your use cases for unsafe Rust?

Participant
Mixed. So sometimes it's just as trivial as a profiling piece of code, and you've got something that loops over vectors, and you just want to get rid of the bounds checks. So it's my code, I know exactly what it does, top to bottom, and it's basically a for loop. It's like, okay, great. I had a hotspot there and I managed to gain an extra 10% performance.

5:8 . So som...
Increase Performance
Profiling

Interviewer
Awesome.

Participant
That's like a trivial case. Another case would be something like, yeah, usually it's like 99% of my use cases actually are just FFI's, either calling things that aren't Rust from Rust or exposing Rust to other things.

5:9 A...
No Other Choice

Interviewer
Gotcha. So let's start with the first category there for performance. So you have these situations you're describing where, I guess just to clarify, you have a Rust application, you're profiling it, and then you notice a particular hotspot.

Participant
In my case, it's almost exclusively libraries, but rather than full applications, but yes.

Interviewer
Oh, gotcha. Okay. So you have a particular library then that you're looking at. You find a hotspot, and then based on the nature of that hotspot, if it involves something like a bounds check that you can eliminate, you introduce unsafe when you are certain that you can do so without causing unsoundness.

Participant
Yeah, pretty much.

Interviewer
Gotcha. Are there any cases aside from bounds checks where you follow this pattern of using unsafe to gain performance?

Participant
Maybe. I haven't thought of one. I think, yeah, another common one is if I

have UTF-8 strings in buffers, but then again, this would be typical of FFI's. So say, for example, I receive a buffer from which I know is in UTF-8, and I trust that the Python interpreter will give me a UTF-8 buffer when I ask for one. I would use a piece of unsafe. I think it's like a UTF-8 unchecked on this stereotype. I forgot what it is, but you basically have a way to elide some checks. So yeah, I mean, if I'm comfortable with the source of my input or with something that couldn't be provable statically to get rid of checks, because some checks like bounds checks that we talked about, some of them you could potentially get rid of at compile time, but not all of them. So if I know better, then I will happily put a little bit of unsafe code there. It depends what you want out of the language. For me, I don't use Rust because I'm particularly paranoid about memory. I've spent all of my life basically dealing with lots of programming languages, and I'm relatively comfortable that certain patterns of code will be safe. So I'm happy with those.

5:11 So say,... Contract or Invariant
Increase Performance

5:10 b... Compiler Elidi...ounds

5:4... Tacit Knowledge

5:12 For me,... Tacit Knowledge

Interviewer
Gotcha. So I guess along those lines, are there any heuristics that you use to determine when it is correct to use unsafe to remove a bounds check?

Participant
Well, if what I'm doing is fully encapsulated, so then that I'm happy to do that.

Interviewer
Gotcha.

Participant
We're now being very specific about that. So if a piece of logic is effectively isolated and encapsulated away, such that what I think are invariants are indeed invariants, then I'm happy to do that.

5:13 We'r... Local, Minimal Unsafe

Interviewer
Gotcha.

Participant
If I think that there's something that may come into the program that will not do that, then yeah.

Interviewer
Gotcha. So I guess to clarify, you have this unsafe code, you have it surrounded in a safe encapsulation. So you have some guarantee because of the safe encapsulation, sort of what your inputs are going to be then, and because of that guarantee you can remove certain checks or use unsafe to increase performance because you know that whatever's
35 coming in is from Safe Rust.

Participant
Would you mind repeating it and then saying yes?

Interviewer
Gotcha. So you mentioned part of the reason why, or the situations where you feel comfortable using unsafe to increase performance. You mentioned that there when you have unsafe code, but it's surrounded in a safe wrapper.

Participant
That's not what I meant. It's not that I meant that it's a safe wrapper. Sure, I will certainly not use the unsafe word beyond what is necessary. So yes, I have a safe wrapper. But obviously at some point I decided that this was the correct boundary layer to delineate the safe wrapper. Now we're talking about why did I decide that marking this code as safe is indeed possible? And that is because I knew that within the program I had certain guarantees that go beyond what could have been expressed by the type system. So for example, yes, this is an integer, but I know that it's never below 1000 and never above 3000. So if the compiler knew that, then it could go off and apply lots of optimizations, but it doesn't. But I do, so there you go. Bye-bye bounds check, because in this case you don't need it. So as simple as that, basically. That's what I mean. So if everything, if I've isolated the code that has the piece of unsafe in a way that I know more about it than the compiler does, then I'm happy to take over.

5:14 Sure, I will certainly not use the uns...

When to Encapsulate?

Interviewer
Gotcha. Yeah, totally following. That makes sense. So then, sorry, I just lost my train of thought. With, oh yeah, so with the case that you gave, maybe that the integer is only between one and 3000, is that something that you find that you need to document for other developers who use those safe APIs? Or are these usually situations where it's within your code and you don't expect other people to be having to follow the right rules to make sure that your code is running safely?

Participant
Sometimes it's just plain obvious what's going on. And that's okay not to put a comment on it, but if it isn't, then yes. There's nothing worse than a comment that is wrong, because now you start wondering, now you start wondering, is it the comment which is wrong, or is the implementation that is wrong? Because comments don't have a compiler. When you chuck enough people at a code base, comments don't really get read all that much. So there's always this thing of, is it worthwhile putting in a comment that may or may not be maintained at this point in time? It's just a pragmatic thing and it's a call you need to make. So I don't actually advocate documenting every piece of unsafe because of that.

5:4...

Audience-Dep...ocume

Interviewer
Gotcha. Interesting. Has there been a particular challenge you've run into

with that where you've encountered a pattern that the community has seen that you feel that is something that should be avoided?

Participant
I think I'm giving you a nugget of experience that I think transcends Rust or not Rust. Just when to comment and when not to comment. So step one, don't comment anything that the code is obviously doing. Step two, if you are going to comment, comment intentionally. Step three, think that this comment may not be maintained as much the risk.

5:16 I think I...

No Docs > Incorrect D

Interviewer
That makes sense. So the second use case for unsafe that you mentioned with foreign functions.

Participant
Yeah, this is by far the more common use case.

Interviewer
Gotcha. So that's generally what you're using. So from your survey responses, it seems like you've used that with quite a few different languages. You have C and C++ as well as Python and Java. Would you describe a bit more about the different situations in which you've needed to use foreign functions?

Participant
Yeah, so I work for a database company and we've got a client for a database and I originally had written this in C. I referred to my previous remarks of not wanting to have a C++ runtime as a dependency and so that went okay. We didn't have any Rust within the company and a requirement came along that this thing had to support TLS and a whole bunch of other authentication stuff on top of that. And next thing you know, I started looking into libraries that do the curves and look at OpenSSL and go, that's not pretty to link. And yeah, I just want this stuff to work like out of the box, like everywhere. So I know Rust and I go, hey, do you mind if we do this in Rust? Please let's not introduce another language. So the whole thing keeps on going for about another four or five weeks and I'm like painfully trying to get this thing working. And in the meantime, I discover a bug in a floating point serialization and some floating point serialization code. So floating point the string, which turns out it's actually a really difficult thing to do accurately. What I mean by that is that if you just take your standard library in C and you do a printf and then you parse that, you're not going to get the same binary representation. So you want to have the minimum number of decimal digits that will give you back the same binary representation that you started with. That turns out that Python does it and a whole bunch of languages do it, Rust does it. So I solved that problem and that takes me another three weeks because of some threading constraints and I try to use the same libraries that Python does. And that turned out to be horribly unproductive. So from then on it's like, okay, fine, we now do this in Rust. 61

5:1...

Databases

What I was then able to do is expose the same C APIs. So that's where the initial bit of FFI was in. So the whole library got rewritten in Rust. It was ABI compatible, which is nice, with the C interface on top. And then obviously that was a C++ interface on top of that, which is where that goes. And then we thought, okay, well, now we have got the C and C++ API. Can we expose the library to Python? So the C API got exposed to Python. And so thinking about the Python library that is basically Python calling Cython, which is just C at the end of the day, which then calls the C API, which then calls the Rust implementation. And now we're basically starting to embed some Rust logic within our core database, which is written in, quote unquote, low latency Java. So we do Java in a way that avoids garbage collection most of the time, basically.

5:19 It was ABI compa...

Directionality

Interviewer
Oh, okay. Gotcha.

Participant
And that's a separate application of Rust from the strings here.

Interviewer
Okay. Gotcha.

Participant
So basically everything that I've done here has been lots of FFI glue.

Interviewer
Gotcha. Yep. I've definitely been there. So you mentioned using cbindgen. Has that been for both of the low latency Java integration with Rust as well?

Participant
JNI.

Interviewer
Oh, JNI. Gotcha.

Participant
Yeah. Yeah. No, because you could, I think there's, it's called JNA, that it sort of dynamically loads the compiled binary and sort of introspects into the available functions and calls them as if it were a C library from Java. Yeah, kind of don't use it because you'll hit a wall in what you can do. And it's not quite as fast as JNI anyways. JNI in itself is already quite slow. You're talking about an order of magnitude slower is like 20, 25 times slower than calling a native Java function.

5:20 Yeah. Yeah. N...

JNA

Limitation of Binding T

Interviewer
Gotcha.

Participant
So yeah, you don't want to make that like 40.

Interviewer
Yeah. Yeah. No, for sure. So with using cbindgen then, or just in, so you would use cbindgen with the serialization library or the string to float, floating point. And then would you write the bindings manually when you're doing it between Rust and JNI?

Participant
Yeah. So within Rust and JNI it's all manual. I've actually hooked up the library for the, so within the client, client got rewritten. It already had a C header. So what I did is that I hooked up cbindgen so that I could double check that what I'm doing is actually the header that I have. Since all the header already had like sections and documentations and macros and things like that, whilst you could replicate most of it in cbindgen, handwritten code just tends to look nicer. It would be nice if there was tooling for both C and Java that tells you whether your FFI conforms to your expectations rather than necessarily tell you what you're expecting generated.

5:21 Yeah. So withi... — Generation VS Validati
5:22 It w... — Wants Solution

Interviewer
Gotcha.

Participant
So do all my symbols match? Do all the types signatures match? Yes, no. What are the differences? And bail out a build and, you know, you could stick it to an NCI or something. That'd be useful. But this code is more than just something that gets compiled in. It's a form of communication. So whenever you have something that you actually want to expose to thousands of people, then it's sort of useful to tweak that so it's nice and readable. Another thing which I don't like about cbindgen is just the compile time. It just adds another like 20 seconds of compile time, at least in my case because I have a library that doesn't actually have all that many dependencies and as soon as I pull cbindgen in it's like 20 seconds every time. So I just made it a feature and I switch it on whenever I want to and then I just diff the header and go from there.

5:44 But thi... — Generation VS Validati
5:23 Another th... — cbindgen

Interviewer
So to confirm, are you using the bindings that are generated or using them as a form of comparison against the ones that you manually write?

Participant
Comparison.

Interviewer

Gotcha. So have you had any issues then? You mentioned wanting a tool that would help reconcile the sort of truth between the bindings you've exposed and what the types actually need to be. Have you had any problems or bugs that you've run into because you've had the bindings declared incorrectly or using a type that's been not quite right?

Participant
Probably during development I must have. It's not like nothing that made it to production that's for sure.

Interviewer
Gotcha.

Participant
So all of these FFI type things for the most part if they go wrong they tend to go horribly wrong and quickly wrong. So ironically that makes it among the least dangerous bugs that you can have. Okay. Because if it blows up then it blows up for good. Have you up there? You can probably end up talking to a security researcher that goes oh yes except for these cases and then you end up with a nice little overflow and blah blah blah and here you go someone took over your system. I'm sort of ignoring that for the best but yeah it's the usual case of write once code review three times.

5:24...
FFI & Binding Bug
Mismatched B...r Decla
5:25 You can pr...
Ignoring Security Issue

Interviewer
Gotcha.

Participant
But yeah if you get it wrong then it'll blow up in your face for the most part. The most serious bugs that I've ever had to deal with are the subtle ones where your program almost produces the correct result.

Interviewer
Okay gotcha.

Participant
Those tend to be the ones that can cost a company.

Interviewer
Do you have any examples of those related to your FFI work with Rust?

Participant
No it's not related to my FFI work. I have plenty of examples of those but not related to my FFI work.

107 **Interviewer**

Gotcha. So in general with the FFI stuff what you've encountered is that when you do make a mistake it's something that is just gonna immediately blow up in your face so that makes it easier to immediately fix them.

Participant
Yeah in the case you either have a linker error or you have an error whenever the symbol is locked up after a deal opens. This is assuming you get the error wrong. It's a little bit more fiddly if you get the signature wrong because now you're basically or if you get like the calling convention wrong then you're basically just sticking the wrong stuff in your registers and your stack before you actually call the function. So that would be harder to diagnose but usually it'll blow up pretty significantly and it's pretty easy to tell when it blows up that it did and then it's just a matter of going oh yeah of course I forgot that and then it's like two seconds to fix.

5:26 Yeah in the case you ei...

FFI & Binding Bug

Mismatched B...r Decla

Interviewer
Gotcha. So have you had any difficulties with the fact that you're working with multiple languages that have different memory models where you have the set of assumptions that Rust makes about its references and then the potentially more flexible assumptions you can have in C and C++? Has that led to any any bugs in your code or any cases of undefined behavior?

Participant
No not bugs but apparently a yes. So it's just like constant oversight of what you're doing. I think one thing which is really cool about Rust is that it has this these marker traits like Send and Sync and things like that. Once you cross an FFI layer those are gone basically so it becomes trivially easy to make concurrency mistakes that you just wouldn't have been able to do if you just stuck to plain vanilla Rust. So that is one thing that maybe is a less obvious thing that you might get bitten at.

5:27 I think one...

Different FFI Memory

Interviewer
Gotcha. So mostly safety of concurrency. Have you run into any issues surrounding like the aliasing and mutability restrictions like having something that Rust assumes is unique and mutable but that actually ends up being like shared and mutated in a bunch of places?

Participant
So the way that I tend to structure code when I do FFI's is that there's generally a clear owner. So usually it's from the mutability point of view like you wouldn't you wouldn't expose fields you expose an API. I haven't really come across that.

5:28 So t...

Simple FFI

Participant
There are some things that are genuinely difficult to to quote-unquote write to unsafe your way out of what I mean by there's so if you have

5:3...

Difficult FFI

Difficult to Encapsulat

methods for example say you have a builder pattern in Rust it's quite common so that you can actually do builders in two flavors in Rust you can either do builders that take self mutate self and then return self so you basically daisy chain until it basically blips out of existence into the thing that you wanted at the end or you could have builders that take a mut self ref reference and then they only destroy the thing at the very end. Writing bindings for builders that take that basically move self along is really hard.

5:31 There are some t...

Difficult FFI

Difficult to Encapsulat

Interviewer
Gotcha.

Participant
So that's one thing that became yeah I had one of those and that became a little bit fiddly.

Interviewer
Could you describe I guess in a bit more detail about what was fiddly about that? In a second.

Participant
No worries. I'll find the code and I'll be able to do that with more.

Interviewer
Awesome.

Participant
With more detail. Yeah so it's just one of those cases where it's it's unsafe code that you need to read like 10 times before you think is this even safe at all. But yeah this is definitely this is definitely something where the the memory model disparity is really obvious.

5:29 Wit...

Unsafe is Diffi...to Und

Interviewer
Gotcha.

Participant
Although I'm looking for this thing.

Interviewer
No worries.

Participant
Yes I think I found I roughly found the code I just need to find the right one. 141

163 **Interviewer**

Gotcha.

Participant
Oh yeah there you go. I'm just going to base it to you because if I start if I start describing it it's going to be properly confusing to even try and describe it.

Interviewer
Gotcha.

Participant
This is... this is a piece of code that you look at and you go what the heck is this doing.

Interviewer
Gotcha.

Participant
That point yeah that pointer read and everything in there.

Interviewer
Interesting. Yeah I'll I'll have to spend some time taking that apart a bit later.

Participant
Yeah good luck. But yeah this is this is because my I'm basically calling a a method on top of a Rust object that takes self but really I'm just mutating it in place.

Interviewer
Oh okay interesting. Gotcha.

Participant
So I'm basically I'm basically completely breaking Rust's expectation because my my C object holds this Rust object a mutable it basically holds a field and then I just want to update it in place as if it were a moot self. But what I'm doing is basically taking it updating it and then putting it back in putting it back in practice so you have this pointer right back into the same location.

5:30 So I'm basical...

Different FFI Memory

Interviewer
Yeah gotcha yeah I think I'm following that yeah no for sure. Okay yeah I I'll I think with with that explanation and having the code is super helpful so I'll be able to think and take that apart a bit more later. That type of example is super awesome for us so thanks a ton for that.

163

Participant
It's really messy.

Interviewer
Yeah I mean that's yeah it's that's sort of the type of thing that we're we're looking for I think finding those cases where there's something that you absolutely need to do in terms of bridging the gap between two different languages and different memory models and where you're you're you're sort of forced into a corner by some of the semantics that Rust has so you need to find a way to get out and how Rust's sort of assumptions about memory and restrictions makes that kind of difficult sometimes. Yeah I guess along those lines just a couple questions you mentioned you use Box and OnceCell in unsafe contexts. Could you describe a bit more about those two?

Participant
Box all the time. Unsafe I don't remember probably I must have picked it so I must have thought during the survey yeah okay oh yeah it was it was OnceCell. I'll oh once I'll have to grab I'll have to grab through my code and check I probably, yeah, Box is, it's, it's pretty common because, so, Box and Rc/Arc they're pretty common. So I either, every time that I create something, so let's say I'm building an FFI I need to create something, the easiest way to to to build the API in a way that is semi-maintainable is to just return a pointer to an opaque object in C. So that means that I'm just going to say type type def struct foo space foo and that's my header and then everything else takes takes a full pointer so at some point I'm going to have it create at some other point I'm going to have a destroy and my create will take a Box and basically my create will take a Box and basically leak it into and across the C FFI and then at some point the destroyer will take that pointer reconstitute it back into a Box and then drop the Box at the very end. So that's typical that's a typical usage of Box across an FFI. Then they for for an Rc/Arc so the Rc/Arc you sort of end up creating a Box of an arc and that is so that you can have an extra layer of indirection say for example that object the object's lifetime exceeds the the frame that I'm calling it from so I may have created it via this create and destroy pair of functions across the C interface or JNI interface or whatever it is the the the the technique is basically the same but at some point the rust code wants to extend the lifetime of that object and so to do that you basically have a root arc that the FFI owns that the foreign client owns and then if you want to extend that then you would do a clone and then that that basically lives this means that even even if even if the client code drops it sorry destroys it cause a destroy function before anything else then it's still alive for the duration of everything else and I think that that double double hop is annoying you could probably like if I really really cared I could probably collapse it down but I would need to have like an intrusive counter or something like that.

5:32 So I...
Opaque Pointer

5:33 my cre...
Allocation in Rust
Box<T>

5:34 Then they for for an Rc/Arc so th...
Allocation in Rust
Arc<T>/Rc<T>
Box<T>

Interviewer
Gotcha.

Participant
You wouldn't be able to do it out of the box with the standard types. You wouldn't be able to do with an Rc/Arc. Okay yeah so you actually have to have a Box of an Rc/Arc it's a bit annoying it's not great for performance

5:35 Y...
Arc<T>/Rc<T>
Box<T>

Interviewer
Gotcha.

Participant
But it's kind of nice yeah so yeah just something you can't get around

Interviewer
So I guess kind of ending on a just a couple broad questions. The first: could you describe a bug that you faced that specifically involved use of unsafe where you were doing some unsafe operation and that in particular was like the source of the bug?

Participant
Certainly not off the top of my head

Interviewer
Gotcha

Participant
I'm thinking there might have been most most bugs ...

Interviewer
Gotcha

Participant
…that make it to production tend not to be because of safety or at least the ones that are identified don't tend to be because of safety they tend to be logical errors so this for example I've had a bug report lately and that was one part of the code requires that all date times are epoch time stamps in in microseconds and they need to be signed integers sorry positive integers I always use unsigned for for daytime for consistency but in one part of the code it had to be you know after unix epoch 1970 january 1st and another case is like well okay in this case it could be negative. So yeah, I validated that always had to be after I validated that that always had to be after 1970 and a customer goes across and says yeah sorry I actually have some old data that I need to put in so that type of bug is far more common than than unsafety bugs those tend to be like they exist but I suspect they they either get detected really really really quickly or they would end up lurking lurking around the system for a long time because they would be they would end up being as part of some not commonly traded code path. Which is where static analysis tooling and

5:36 that make it to pr...
Logical Error
Requirements Bug

things like that and compilers really being pedantic about these things help

Interviewer
Yeah

Participant
Because the standard methods of testing don't tend to pick up on them.

Interviewer
Gotcha, and then speaking of standard testing methods, do you feel that your current development tools handle all the problems you face when writing unsafe or are there areas where you feel that your current tools could be improved or a new tool would be helpful for you so like I'm thinking like the example you gave earlier of of something that can automatically reconcile the differences between the foreign bindings that you're creating and what you actually need them to do...

Participant
Yeah bindings is always a fiddly mess and the reason for that is that you will inevitably have to at the very least duplicate signatures in two languages at the very least usually it's at the very least usually it's at least three often four times because you tend in practice what you tend to have is if say you're you're bridging language A and language B you would tend to have a high level implementation in language A a low level binding in language A for language B a low level implementation in language B and then a high level implementation in language B so because of that you basically have the same thing slightly different four times keeping those four versions of it in sync is often but not always aided between the tooling

5:37 Yeah bindings is always a f...

Bindings are a Fiddly

Interviewer
Gotcha

Participant
So yeah there's always room for improvement there which is why you look at projects like bindgen, cbindgen, CXX and blah blah blah at least in the Rust space, and also why you look at things in the Java space and you have like JNI, JNA and i think there's like new project whatever I'm not going to use it because i still need to support all JVM's thing yeah and then in Python you've siphoned C types, boost python yeah like there's like and libraries to to do the same thing and they all have their weak spots. So yeah my bindings are fiddly.

Interviewer
So i think that's pretty much all the questions that i had today you know so you're just saying something yeah

Participant
I think your your question was a little bit more generic than just um than just bindings um yeah i mean there's there's there's two things here there's there's obviously the the tooling aspect there's also the language itself aspect so there's there's cases where it would be nice to potentially maybe like slightly more formally document certain assumptions certain invariants or things like that um so I have code for example where where i use unsafe during um so I'm building from raster dynamic library um a whole bunch of unsafe wrappers for things which are only ever set once when the library is loaded so i kind of don't want to have locks afterwards so that's you know that's another use case of unsafe and you know does it really have to be unsafe isn't there a better isn't there a pattern around this so that i don't need to care about

5:38 I think your your question was...

Wants Solution

Interviewer
Yeah, yeah, gotcha.

Participant
So tooling sometimes is inside language sometimes it's outside sometimes it's library sometimes it's yeah and and with tooling in general uh you have things like uh fsanitize and then you have Valgrind and things like that and then half of the battle with tooling is just um it's just a ease of use and and how integrated it comes. So yeah i'll i'll be honest half time half the time i won't use a tool just because it'll take me so long to set up and i need to tweak the kernel parameters just to get it running and maybe I'm currently not on linux and good luck getting anything working on on those and and honestly just getting gdb working on the on on a mac is a pain and hassle in itself because lldb will work just fine but honestly i never remember all the arguments to it so yeah half of the time you'll just go okay can i just stare at this for another five minutes grab a coffee and then is that good enough so programmer laziness is um is yeah fights tooling unfortunately

5:39 So yeah i'll i'll be honest h...

Laziness Fights Toolin

Participant
Can i ask uh yeah just what you're doing i'm assuming this is some related to your PhD project are you working in a group what are you what are you trying to accomplish because yeah i'm curious.

Interviewer
Yeah, yeah, so um yeah so my my PhD is now centered on unsafe rust and i'm at the point where i sort of have two approaches one is a more formal methods approach thinking about okay how can we expand rust type system to prove maybe some more of the invariance you were just talking about that would be useful for you but then i think another aspect of this is that there are a little for logic inside there and yeah exactly yeah um that'll be fun yeah and then i think part of it though is you can write greek and complex formulas all day long but none of that's really going to get you anywhere if you don't understand what the current problems are

that people are facing and there isn't a bunch of research on that specific to unsafe rust there's a lot of what do people struggle with when they're writing rust but there's not a lot of specifically like when you're using unsafe what are the issues you're facing so the goal of this study is to speak to a inordinately large amount of rust developers as many people as i can find um and hear about all of the different perspectives so then i can write one big summary that goes through just the challenges everyone has so that it can be like okay all right here are like the maybe five key areas i don't know if they're going to be five or not i'm just pulling that number but here are the five key challenges maybe that are generally shared among all unsafe developers and here are all of the ways that the current research community could try to create solutions for them so that's sort of the goal of this study in particular.

Participant
Yeah there's there's actually one other problem that I forgot to mention, and I think this is genuinely a problem within, uh, within unsafe code, and that is lies. Um, what I mean by that is that sometimes you have safe wrappers to things that aren't safe, and actually this is really really really common, um, especially when you have libraries, Rust libraries that wrap, say, a C API, and to make it usable, it'll make a bunch of assumptions but those set of assumptions aren't actually correct, true, and valid. So now, it basically, so now it basically, it, it basically created a lie, quote unquote, that this is safe, when in actual fact, um, it isn't.

5:40 Yeah there's there's ac...

"Safe" API

Interviewer
Gotcha

Participant
So that is actually, I think I see why people do it. It's kind of necessary, um, and there's sort of no way around that, as there's no, no way to, to express this something as being, because it's like, you just infinitely label it as unsafe, um, and it's, yeah, like a standard thing would be like, something that loads up something as a memory map file, um, and then halfway through the C library just unmaps the file, but, and it gives, you have a reference to something that you thought was good, and then the whole thing dies.

5:45 So that is actuall...

"Safe" API

Difficult to Encapsulat

Interviewer
Gotcha

Participant
So, you know, how do you guard against it? And it's like, well, usually it works, but the original, the original author of the Rust library thought it would work, but you, you basically have, you know you, you have no guarantees, so it's a trust layer, but sometimes those, those, the, the, the trust layer is, is buggy.

5:41 So, you...

"Safe" API

Interviewer

Gotcha that makes sense so yeah hey they know that that yeah that that lines up with a lot of what you were describing to earlier with the FFI stuff in terms of who do you, like, who is responsible, where where does the trust fall?

251