



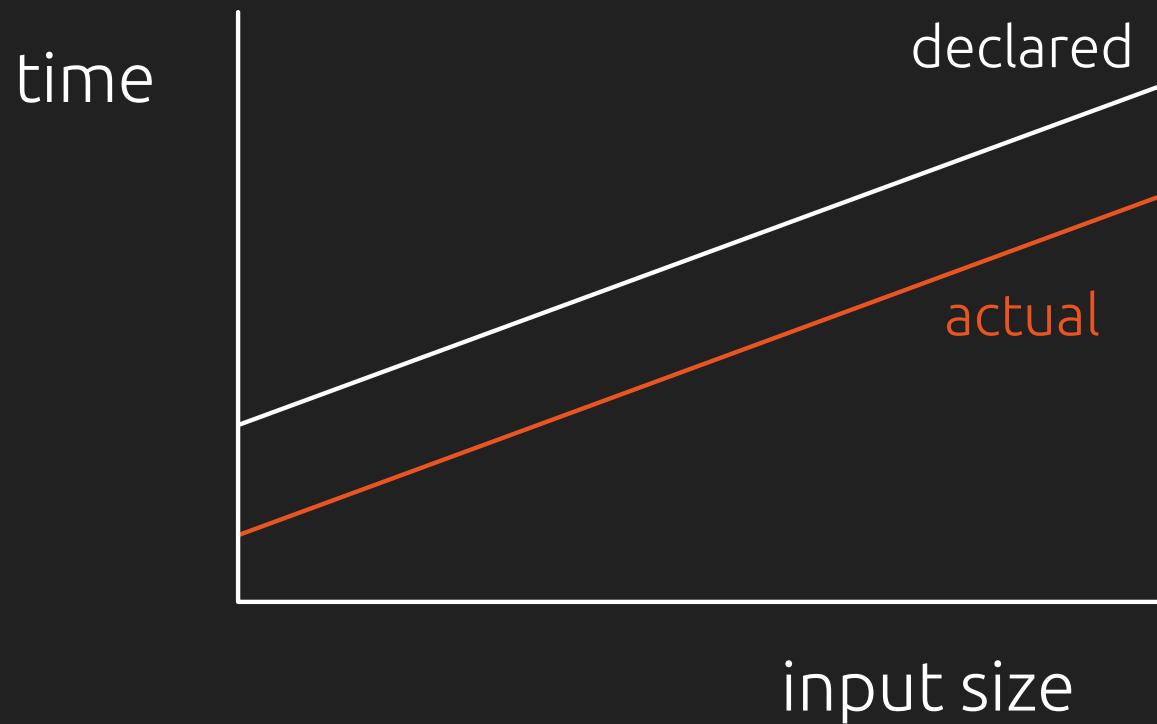
# Fearless Refactoring and the Art of Argument-Free Rust

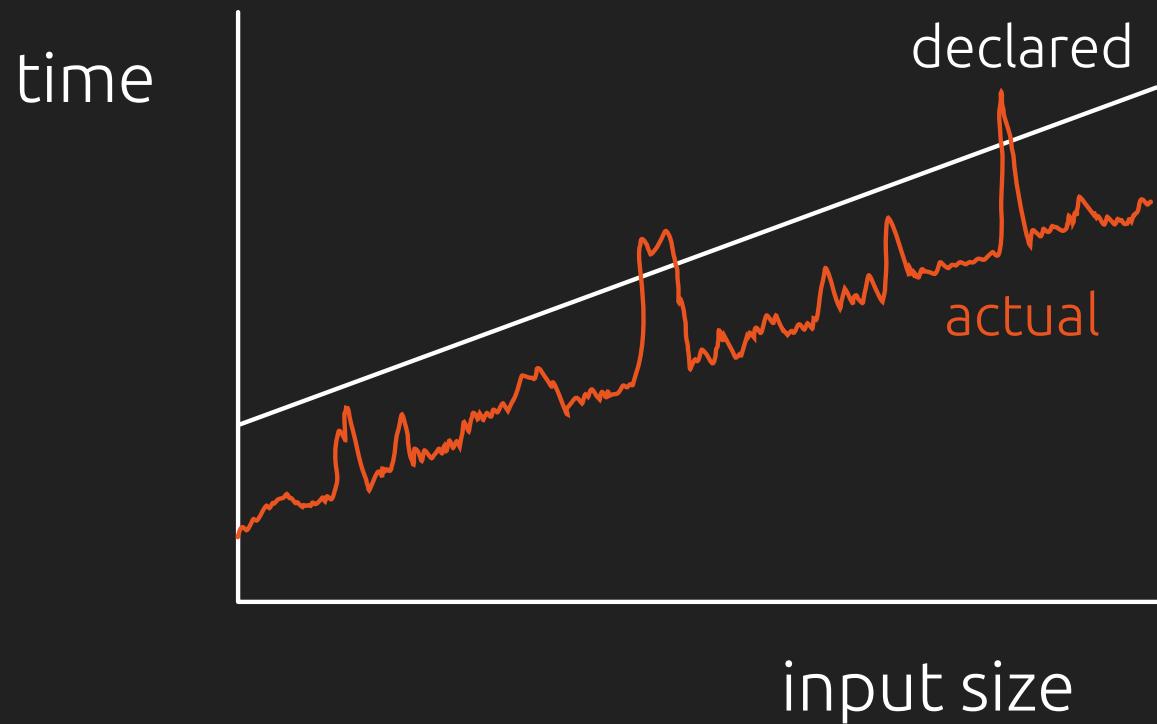
*Ed Jones, RustConf '24*

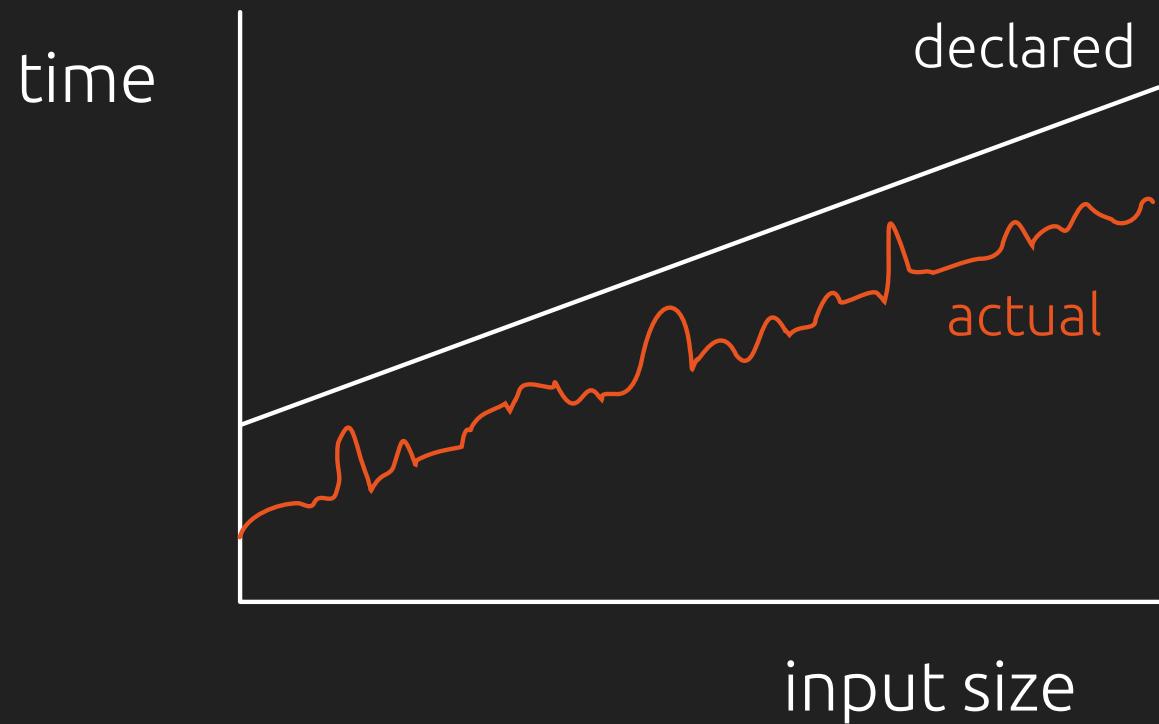
- I work at Canonical

- I work at Canonical
- I like clean code

- I work at Canonical
- I like clean code
- I annoy my coworkers







industry's hard problems

# industry's hard problems

- requirements

# industry's hard problems

- requirements
- politics

# industry's hard problems

- requirements
- politics
- maintenance

what's maintainable?

# what's maintainable?

- easily-described concepts

# what's maintainable?

- easily-described concepts
- cleanly-flowing concepts

# what's maintainable?

- easily-described concepts
- cleanly-flowing concepts
- predictable patterns

for whom does the coder code?

# for whom does the coder code?

- consumers

# for whom does the coder code?

- consumers
- coworkers

# for whom does the coder code?

- consumers
- coworkers
- themselves

# fearless refactoring

# fearless refactoring

after making a change, the next time all checking passes,  
code likely still works as intended

to help the refactoring:

to help the refactorer:

- avoid `unwrap`

# to help the refactorer:

- avoid `unwrap`
- match exhaustively

# to help the refactorer:

- avoid `unwrap`
- match exhaustively
- mutate in small scopes

# to help the refectorer:

- avoid `unwrap`
- match exhaustively
- mutate in small scopes
- limit unsafety

# to help the refectorer:

- avoid `unwrap`
- match exhaustively
- mutate in small scopes
- limit unsafety
- be sparingly weird

weirdness

# weirdness

- is equivalent to surprise

# weirdness

- is equivalent to surprise
- diverts attention

# weirdness

- is equivalent to surprise
- diverts attention
- is resolved by debate

# The Art of Argument-free Rust

# The Art of Argument-free Rust

*note: not debate-free*

fostering debate

# fostering debate

- cross-team code review

# fostering debate

- cross-team code review
- office hours ('Rust Surgery')

# fostering debate

- cross-team code review
- office hours ('Rust Surgery')
- standardisation docs

## General definition ordering

When read from top to bottom, a file should feel like a tour of the APIs it defines. The most important items should be defined further up the file, with their helpers below. No `impl` block should come before the type or trait to which it relates. In this way, lower-level implementation details are hidden from the reader until they wish to know more, at which point, having gained a good knowledge of the overall form of the code, they can read on to understand how it functions.

 Do this:

```
impl Foo {
    pub fn some_func(&self) {
        self.some_helper_func();
    }

    fn some_helper_func(&self) {
        // ...
    }
}
```

 Avoid this:

```
impl Foo {
    fn some_helper_func(&self) {
        // ...
    }

    pub fn some_func(&self) {
        self.some_helper_func();
    }
}
```

- [Error and panic discipline](#)
  - [Error messages](#)
  - [Error types](#)
  - [Error conversion](#)
  - [Panic calmly](#)
- [Function discipline](#)
  - [No-information returns](#)
  - [Hide generic type parameters](#)
  - [Unused parameters default implementations](#)
  - [Builder visibility](#)
  - [Builder ownership](#)
- [Ordering discipline](#)
  - [General definition ordering](#)
  - [Impl block placement](#)
  - [Impl block ordering](#)
  - [Derive ordering](#)
  - [Declaration ordering](#)
  - [Struct field ordering](#)
- [Unsafe discipline](#)
  - [Minimise unsafe](#)
  - [Document preconditions](#)
- [Structural discipline](#)
  - [How to structure `mod.rs`](#)
  - [Use `mod.rs` to declare a module-root](#)
  - [Define `Error` and `Result` in a standard location](#)

the process

# the process

1. note repeated arguments

# the process

1. note repeated arguments
2. write up generally-applicable notes

# the process

1. note repeated arguments
2. write up generally-applicable notes
3. review with a few experienced Rustaceans

# the process

1. note repeated arguments
2. write up generally-applicable notes
3. review with a few experienced Rustaceans
4. review with many Rustaceans

# the process

1. note repeated arguments
2. write up generally-applicable notes
3. review with a few experienced Rustaceans
4. review with many Rustaceans
5. release

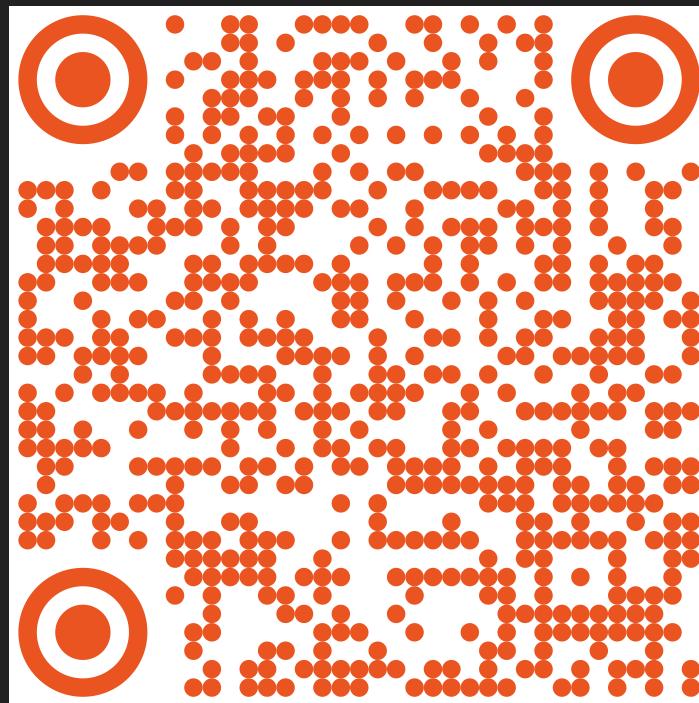
# the process (hard-mode)

1. note repeated arguments
2. write up generally-applicable notes
3. review with a few experienced Rustaceans
4. review with many Rustaceans
5. release

the process (easy-mode)

# the process (easy-mode)

[github.comcanonical/rust-best-practices](https://github.comcanonical/rust-best-practices)



what's next?

# what's next?

- more standards

# what's next?

- more standards
- automation

# what's next?

- more standards
- automation
- open office hours

thanks for listening!

