```
$ cat ~/.profile
GIT_AUTHOR_NAME=Florian Gilcher
GIT_AUTHOR_EMAIL=florian@asquera.de
TWITTER_HANDLE=argorak
GITHUB_HANDLE=skade
BLOG=skade.me
YAKS=yakshav.es
```

- Rust and Elasticsearch Trainer
- Event organiser
- Ruby Programmer since 2003
- Rust Programmer since 2013
- CEO asquera GmbH

- Community person
- Rust/Search Meetups
- eurucamp/jrubyconf.eu
- RustFest
- Part of the global Rust community team

As a hobby, I shoot arrows at stuff

DKyuB

Deutscher Kyudo Bund e.V.

German Kyudo Federation

Why is Rust successful?

Problem

There’s almost no comparable metrics for programming languages around.

Define "Success"

- Moves the state of technology forward
- Is used in sizable production environments
- Attracts contribution
- Has sizable growth
- Attracts positive feedback

What is Rust?

- new systems programming language
- powers and was developed in along with Servo, a new browser engine
- by Mozilla and the Community
- First stable release May 15th, 2015

Providing an alternative to C/C++, but also higher-level languages.

- Safe
- Concurrent
- Fast

It’s generally perceived that safety, especially memory-safety comes at a runtime cost.

- Safe
- Concurrent
- Fast

Pick Three

Core features

- Static type system with local type inference
- Explicit mutability
- Zero-cost abstractions
- Runtime-independent concurrency safety

- Errors are values
- No null
- Static automatic memory management
- No garbage collection

```rust
extern crate tempdir;

use tempdir::*;
use std::fs::File;

fn main() {
    let tempdir = TempDir::new("goto-berlin");

    let mut tempfile = match tempdir {
        Ok(dir) => {
            File::create(
                dir.path().join("tmpfile")
            )
        }
        Err(_) => { panic!("Couldn't open tempdir") }
    }

    do_something(&mut tempfile);

    // look, no close necessary!
}
```

Base concept: Mutability

```rust
struct InnerData {
     val: i32
}

struct Data {
    inner: InnerData
}

fn main() {
    let d = Data { inner: InnerData { val: 41 }};
    d.inner.val = 42;
    // error: cannot assign to immutable field `d.inner.val`
}
```

```rust
struct InnerData {
     val: i32
}

struct Data {
    inner: InnerData
}

fn main() {
    let mut d = Data { inner: InnerData { val: 41 }};
    d.inner.val = 42;
}
```

Base concept:
Ownership & Borrowing

- Every piece of data is uniquely owned
- Ownership can be passed
- When owned data reaches the end of a scope, it is destructed

```rust
use std::fs::File;
use std::io::Write;

fn main() {
    let file = File::open("test")
        .expect("Unable to open file, bailing!");
    take_and_write_to_file(file);

    // take_and_write_to_file(file);
    // ^^ Illegal
}

fn take_and_write_to_file(mut file: File) {
    writeln!(file, "{}", "Hello #gotober!");
}
```

- Access can be borrowed (mutable and immutable)
- You can borrow mutably once
- Or multiple times immutably
- Exclusive: mutable or immutable, never both

Shared mutable state is an issue even single-threaded applications!

```rust
use std::fs::File;
use std::io::Write;

fn main() {
    let mut file = File::open("test")
        .expect("Unable to open file, bailing!");

    write_to_file(&mut file);
    write_to_file(&mut file);
}

fn write_to_file(file: &mut File) {
    writeln!(file, "{}", "Hello #gotober!");
}
```

```rust
fn main() {
    let mut vector = vec![1,2,3];
    let elem = &vector[1];
    vector[2] = 4;
}
```

```
error[E0502]: cannot borrow `vector` as mutable
 --> src/main.rs:4:5
  |
3 |     let elem = &vector[1];
  |                 —— immutable borrow occurs h
4 |     vector[2] = 4;
  |     ^^^^^^ mutable borrow occurs here
5 | }
  | - immutable borrow ends here
```

Rust checks validity of all references at compile-time.

```rust
struct Data<'a> {
    inner: &'a i32
}

fn return_reference<'a>() -> Data<'a> {
    let number = 4;

    Data { inner: &number }
}
```

```
 --> src/main.rs:8:20
  |
8 |     Data { inner: &number }
  |                   ^^^^^^^ does not live long
9 | }
  | - borrowed value only lives until here
  |
```

All Rust function signatures not only signal data types, but also mutability, ownership and interconnections between input and output types.

100, 1000, 10.000 lines of called code, Rust keeps these properties!

Abstractions

Rust provides higher-level abstractions through Generics and Traits, similar to C++ Templates or Java Generics.

Concurrency without fear

```rust
let counter = Counter { count: 0 };

for _ in 1..3 {
    std::thread::spawn(move || {
        increment(&mut counter);
        // capture of moved value: `counter`
    });
}
```

```rust
use std::rc::Rc;
let rc = Rc::new(Counter { count: 0 });

for _ in 1..3 {
    let handle = rc.clone();
    std::thread::spawn(move || {
        // `std::rc::Rc<Counter>` cannot be sent between threads safely
        increment(&mut handle);
    });
}
```

```rust
use std::rc::Rc;
let rc = Rc::new(Counter { count: 0 });

for _ in 1..3 {
    let handle = rc.clone();
    std::thread::spawn(move || {
        increment(&mut handle.lock().unwrap());
    });
}
```

This example could be a concurrency bug in many languages, or even a double-free!

This analysis is purely static and independent of concurrency primitive! Rusts type system allows no data races.

Low-level control & safety!

- Borrows boil down to pointers at runtime
- Values are plain values just like in e.g. C
- Optional unsafe sub-language

“ Safe code means you can take better risks.”
– @QEDunham

Is Rust successful?

Moves the state of technology forward

Rust wraps safety techniques previously only used in research settings in a production-ready package. It brings memory-safety to environments where it was previously not possible.

Is used in sizable production environments

- Dropbox has sizable backend systems in it
- Chef Habitat is written using Rust
- Canonical: from server monitoring to middleware
- Used in several games
- Schauspiel Dortmund

https://www.rust-lang.org/friends.html

Attracts contribution

More then 4/5 of contributions
to the Rust language come
from outside Mozilla.
The compiler has more
then 2000 contributors.

https://thanks.rust-lang.org/rust/all-time

GitHub

01

02

03

04

Projects with the most reviews

DEFINITELYTYPED/DEFINITELYTYPED	800
KUBERNETES/KUBERNETES	680
HOMEBREW/HOMEBREW-CORE	580
ANSIBLE/ANSIBLE	550
NODEJS/NODE	480
NIXOS/NIXPKGS	480
APACHE/SPARK	450
RUST-LANG/RUST	390
SYMFONY/SYMFONY	340
TENSORFLOW/TENSORFLOW	340

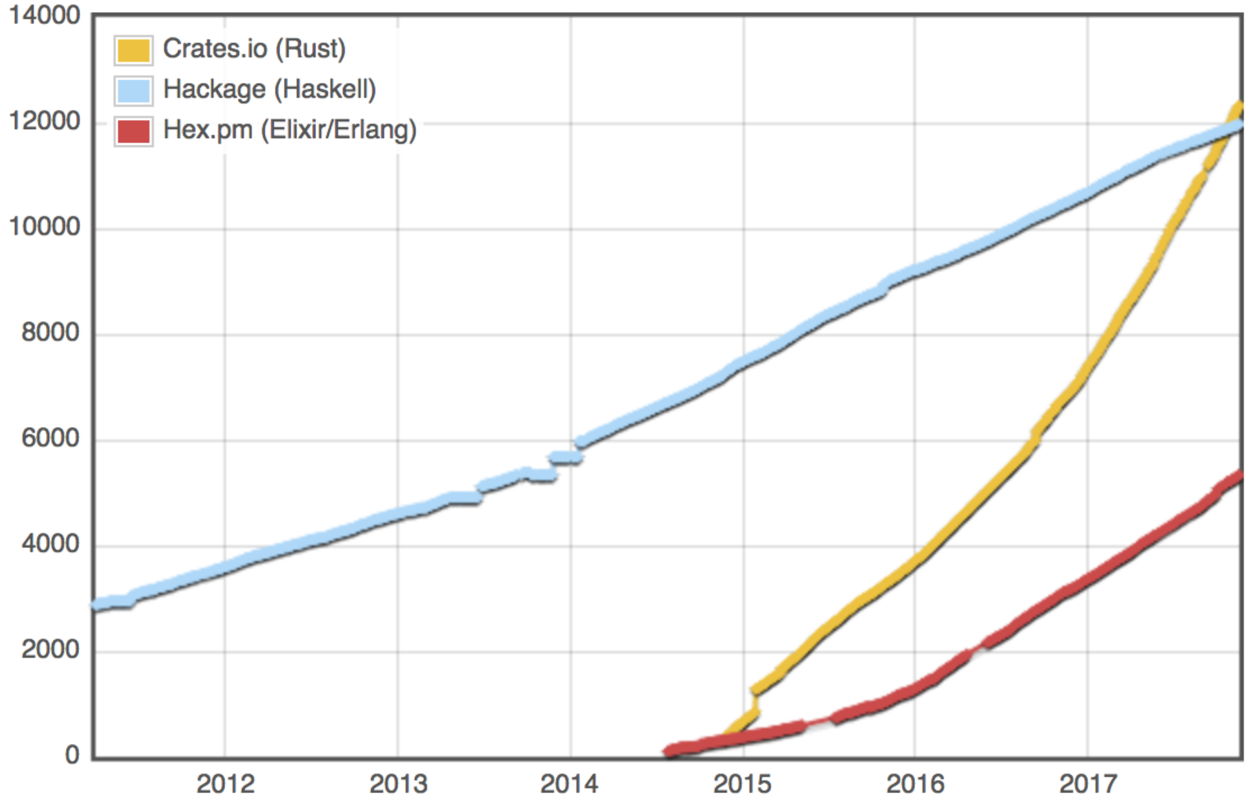
Ten most-discussed repositories

KUBERNETES/KUBERNETES	388.1K
OPENSHIFT/ORIGIN	91.1K
CMS-SW/CMSSW	80.1K
MICROSOFT/VSCODE	78.7K
RUST-LANG/RUST	75.6K
DOTNET/COREFX	75.2K
TGSTATION/TGSTATION	74.8K
NODEJS/NODE	66.3K
SERVO/SERVO	54.9K
ANSIBLE/ANSIBLE	53.9K

Has sizable growth

- Doubled in TIOBE and Redmonk indices since last year.
- This maps to increased survey respondent numbers
- Currently has 5 conferences per year
- Over 100 Meetups around the world
- Newcomers from all directions

Module Counts



Attracts positive Feedback

- Voted "Most loved language" on Stackoverflow 2017, 2016
- Only 1/100 of survey respondents have issues with the Rust community

117

Bless you Rustaceans! (self.rust)

eingereicht vor 12 Stunden von daveburnt

10 Kommentare Weitersagen Speichern Ausblenden melden crosspost

Why is Rust successful?

Rust wants to be used

Rust was always intended as a
industry production language.
For that reason, it was always
developed in lockstep with
a huge application (Servo).

Rust is approachable

- Easy to get an environment setup
- Build and dependency management part of the toolchain
- Development tooling as well, like linters
- We take pride in error messages

- Great docs through the book and a (almost) fully documented standard API
- Central YouTube channel with talks and lectures
- Maintained forums and IRC

Rust is pragmatic

- All features are driven by needs of real software
- The language is evolved with user feedback
- Consciously chooses familiar constructs
- Picks good ideas from others

Rust plays well with others

- Rust interfaces to other languages through the C ABI at zero cost
- Binding generators C and C++
- High-level bindings for some languages (Ruby, JavaScript)

Rust is a great partner

- Preserves the conveniences of high-level languages in low-level land.
- Doesn’t want to be the primary language at all cost
- Built for gradual adoption

Project: Stylo

Stylo is Firefox new styling engine, taken from Servo.

- Stylo is heavily parallel and concurrent
- Multithreading without support was a huge source of bugs
- Concurrency assistance allowed a rewrite in about a year
- Rusts FFI allowed easy integration

Beyond plain code

Commitment to stability

- Rust is released every 6 weeks
- Rust is backwards-compatible, with huge machinery to ensure that
- Currently at version 1.21.0

Commitment to stability

- Rust stable allows no use of unstable features
- These are only available in nightly builds
- Features that are not ready for wide use are not released for wide use.

Maturity

- Code generation is provided through LLVM
- No runtime, no runtime bugs
- Very conservative approach to stdlib adoption

Cross-capabilities

- rustc is a cross-compiler by default and the whole toolchain is aware
- Almost-no-setup cross compilation! (getting better)
- Rust supports embedded and IoT usecases
- Yes, we WASM

Governance & Care

All changes go through
an open RFC process!

Mozilla has great experience with open discussion and processes and it shows!

Medium time to merged PR for changes: 6 days!

Supporting commercial users

- regular interviews with prod users, to hear about their issues
- the core team can always be spoken to
- consulting, development and training available, through integer32 and asquera.

Have a question?
community-team@rust-lang.org

Rust brings safe programming to targets where it was unfeasible before while also bringing new things to the table to compete with other safe languages.

You can’t spell trust without Rust