WORKSHOP TOPICS

- Testing setup in Rust
- Writing unit tests
- Writing integration tests
- Integration tests vs functional tests
- Documentation tests
- Test doubles: traits vs Mocks
- Test isolation
- Exploratory testing
- The Rust testing ecosystem
- Speeding up tests
- Advanced testing strategies

TESTING SETUP IN RUST

CARGO IS RUST'S SWISS ARMY KNIFE

```
cargo --help

Commands:
    build, b    Compile the current package
    check, c    Analyze the current package and report errors
    clean       Remove the target directory
    doc, d      Build this packages and its dependencies documentation
    new         Create a new cargo package
    init        Create a new cargo package in an existing directory
    add         Add dependencies to a manifest file
    remove      Remove dependencies from a manifest file
    run, r      Run a binary or example of the local package
    test, t     Run the tests
    bench       Run the benchmarks
    update      Update dependencies listed in Cargo.lock
    search      Search registry for crates
    publish     Package and upload this package to the registry
    install     Install a Rust binary
    uninstall   Uninstall a Rust binary
```

corrode

CARGO IS THE TOOL FOR RUNNING TESTS

```
cargo --help

Commands:
    build, b    Compile the current package
    check, c    Analyze the current package and report errors
    clean       Remove the target directory
    doc, d      Build this packages and its dependencies documentation
    new         Create a new cargo package
    init        Create a new cargo package in an existing directory
    add         Add dependencies to a manifest file
    remove      Remove dependencies from a manifest file
    run, r      Run a binary or example of the local package
    test, t     Run the tests
    bench       Run the benchmarks
    update      Update dependencies listed in Cargo.lock
    search      Search registry for crates
    publish     Package and upload this package to the registry
    install     Install a Rust binary
    uninstall   Uninstall a Rust binary
```

corrode

UNIT TESTS

UNIT TESTS ARE NEXT TO CODE

```
1  > pub fn is_prime(n: u64) -> bool {···
11    }
12
13    #[cfg(test)]
14  > mod tests {···
30    }
```

corrode

SIMPLE EXAMPLE CODE

```
pub fn is_prime(n: u64) -> bool {
    if n < 2 {
        return false;
    }
    for i: u64 in 2..n {
        if n % i == 0 {
            return false;
        }
    }
    true
}
```

corrode

UNIT TESTS

```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_is_prime() {
        assert_eq!(is_prime(0), false);
        assert_eq!(is_prime(1), false);
        assert_eq!(is_prime(2), true);
        assert_eq!(is_prime(3), true);
        assert_eq!(is_prime(4), false);
        assert_eq!(is_prime(5), true);
        assert_eq!(is_prime(6), false);
        assert_eq!(is_prime(7), true);
        assert_eq!(is_prime(8), false);
        assert_eq!(is_prime(9), false);
    }
}
```

corrode

RUNNING UNIT TESTS

```
~/C/c/w/testing >>> cargo test
   Compiling testing v0.1.0 (/Users/mendler/Code/corrode/workshops/testing)
    Finished test [unoptimized + debuginfo] target(s) in 0.28s
     Running unittests src/lib.rs (target/debug/deps/testing-12004ce4ce0e713d)

running 1 test
test tests::test_is_prime ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

   Doc-tests testing

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

UNIT TESTS

```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_is_prime() {
        assert_eq!(is_prime(0), false);
        assert_eq!(is_prime(1), false);
        assert_eq!(is_prime(2), true);
        assert_eq!(is_prime(3), true);
        assert_eq!(is_prime(4), false);
        assert_eq!(is_prime(5), true);
        assert_eq!(is_prime(6), false);
        assert_eq!(is_prime(7), true);
        assert_eq!(is_prime(8), false);
        assert_eq!(is_prime(9), false);
    }
}
```

corrode

UNIT TESTS

```rust
#[test]
fn test_is_prime() {
    let cases: Vec<(u64, bool)> = vec![
        (0, false),
        (1, false),
        (2, true),
        (3, true),
        (4, false),
        (5, true),
        (6, false),
        (7, true),
        (8, false),
        (9, false),
    ];

    for (n: u64, expected: bool) in cases {
        assert_eq!(is_prime(n), expected);
    }
}
```

corrode

UNIT TESTS

```rust
#[test]
fn test_is_prime() {
    let primes: [u64; 5] = [2, 3, 5, 7];

    for n: &u64 in primes.iter() {
        assert!(is_prime(*n));
    }
}

#[test]
fn test_is_not_prime() {
    let not_primes: [u64; 6] = [0, 4, 6, 8, 9, 10];

    for n: &u64 in not_primes.iter() {
        assert!(!is_prime(*n));
    }
}
```

corrode

UNIT TESTS

```
#[test]
fn test_is_prime() {
    let primes: [u64; 4] = [2, 3, 5, 7];

    for n: u64 in primes.into_iter() {
        assert!(is_prime(n));
    }
}

#[test]
fn test_is_not_prime() {
    let not_primes: [u64; 6] = [0, 4, 6, 8, 9, 10];

    for n: u64 in not_primes.into_iter() {
        assert!(!is_prime(n));
    }
}
```

corrode

INTEGRATION TESTS

“

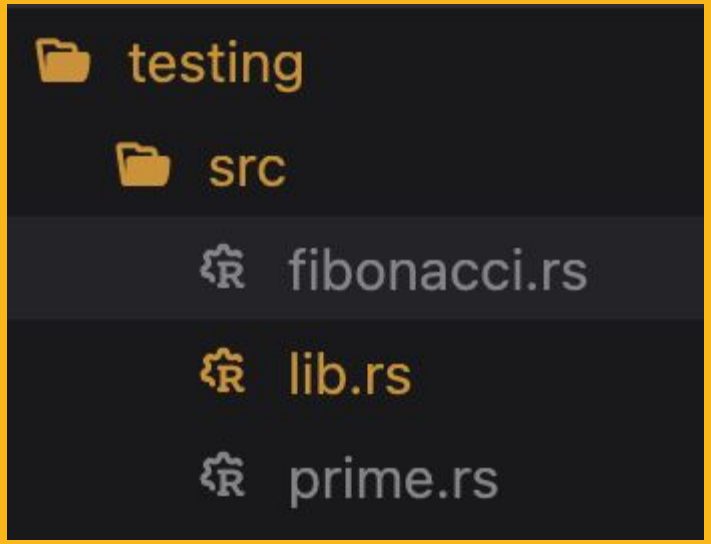
> TO TEST THE FUNCTIONALITY OF A GROUP OF DIFFERENT MODULES AND UNDERSTAND HOW THEY INTERACT WITH EACH OTHER

Definition Integration Test

corrode

https://www.practitest.com/resource-center/article/functional-test-vs-integration-test/

LET'S EXTEND OUR FILE TREE



corrode

LET'S ADD ANOTHER FUNCTION

```
pub fn fibonacci(n: u64) -> u64 {
    if n == 0 {
        return 0;
    }
    if n == 1 {
        return 1;
    }
    let mut a: u64 = 0;
    let mut b: u64 = 1;
    let mut c: u64 = 0;
    for _ in 2..=n {
        c = a + b;
        a = b;
        b = c;
    }
    c
}
```

corrode

MAKE MODULES PUBLIC

src/lib.rs

```rust
pub mod fibonacci;
pub mod prime;
```

corrode

TESTS/INTEGRATION.RS

tests/integration.rs

```rust
use testing::fibonacci::fibonacci;
use testing::prime::is_prime;
```

corrode

TESTS/INTEGRATION.RS

tests/integration.rs

```rust
use testing::fibonacci::fibonacci;
use testing::prime::is_prime;

#[test]
fn test_fibonacci_is_prime() {…
}
```

corrode

TESTS/INTEGRATION.RS

tests/integration.rs

```rust
use testing::fibonacci::fibonacci;
use testing::prime::is_prime;

#[test]
fn test_fibonacci_is_prime() {
    let fib_is_prime: [(u64, u64, bool); 4] = [
        (0, 0, false),
        (1, 1, false),
        (2, 1, false),
        (3, 2, true),
    ];

    for (n: u64, fib: u64, prime: bool) in fib_is_prime.into_iter() {
        dbg!(n, fib, prime);
        assert_eq!(fibonacci(n), fib);
        assert_eq!(is_prime(fibonacci(n)), prime);
    }
}
```

corrode

CARGO TEST

```
~/C/c/w/testing >>> cargo test
   Compiling testing v0.1.0 (/Users/mendler/Code/corrode/workshops/testing)
    Finished test [unoptimized + debuginfo] target(s) in 0.38s
     Running unittests src/lib.rs (target/debug/deps/testing-12004ce4ce0e713d)

running 3 tests
test fibonacci::tests::test_fibonacci ... ok
test prime::tests::test_is_prime ... ok
test prime::tests::test_is_not_prime ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

     Running tests/integration.rs (target/debug/deps/integration-527b3a6d16ab00ec)

running 1 test
test test_fibonacci_is_prime ... ok
```

corrode

CARGO TEST

```
~/C/c/w/testing >>> cargo test
   Compiling testing v0.1.0 (/Users/mendler/Code/corrode/workshops/testing)
    Finished test [unoptimized + debuginfo] target(s) in 0.38s
     Running unittests src/lib.rs (target/debug/deps/testing-12004ce4ce0e713d)

running 3 tests
test fibonacci::tests::test_fibonacci ... ok
test prime::tests::test_is_prime ... ok
test prime::tests::test_is_not_prime ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

     Running tests/integration.rs (target/debug/deps/integration-527b3a6d16ab00ec)

running 1 test
test test_fibonacci_is_prime ... ok
```

corrode

INTEGRATION TESTS
VS
FUNCTIONAL TESTS

corrode

“

> TO TEST THE FUNCTIONALITY OF A GROUP OF DIFFERENT MODULES AND UNDERSTAND HOW THEY INTERACT WITH EACH OTHER

Definition Integration Test

corrode

https://www.practitest.com/resource-center/article/functional-test-vs-integration-test/

“

TO VERIFY THE SOFTWARE IS FUNCTIONING AS IT SHOULD IN A REAL-LIFE SITUATION, AND ALSO MEETS THE USER REQUIREMENTS

Definition Functional Test

https://www.practitest.com/resource-center/article/functional-test-vs-integration-test/

corrode

FUNCTIONAL TESTS IN RUST

- Live in the `tests/` folder
- Same semantics as integration tests
- But the focus is on real-world scenarios

DOCUMENTATION TESTS

CARGO IS RUST'S SWISS ARMY KNIFE

```
cargo --help

Commands:
    build, b    Compile the current package
    check, c    Analyze the current package and report errors
    clean       Remove the target directory
    doc, d      Build this packages and its dependencies documentation
    new         Create a new cargo package
    init        Create a new cargo package in an existing directory
    add         Add dependencies to a manifest file
    remove      Remove dependencies from a manifest file
    run, r      Run a binary or example of the local package
    test, t     Run the tests
    bench       Run the benchmarks
    update      Update dependencies listed in Cargo.lock
    search      Search registry for crates
    publish     Package and upload this package to the registry
    install     Install a Rust binary
    uninstall   Uninstall a Rust binary
```

corrode

RUNNING CARGO DOC

testing — zsh

```
~/C/c/w/testing ❯❯❯ cargo doc --document-private-items --no-deps --open
```

corrode

RUNNING CARGO DOC

testing
0.1.0

All Items

Modules

Crates
testing

Click or press 'S' to search, '?' for more options...

?

Crate testing

source · [-]

Modules

fibonacci
prime

corrode

RUNNING CARGO DOC

testing
0.1.0

In
testing::prime

Functions

is_prime

Click or press 'S' to search, '?' for more options...

?

Function testing::prime::is_prime

source · [-]

```
pub fn is_prime(n: u64) -> bool
```

corrode

DOCS?

corrode

OUR PREVIOUS `is_prime` EXAMPLE

src/prime.rs

```rust
pub fn is_prime(n: u64) -> bool {
    if n < 2 {
        return false;
    }
    for i: u64 in 2..n {
        if n % i == 0 {
            return false;
        }
    }
    true
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_is_prime() {
        let primes: [u64; 4] = [2, 3, 5, 7];

        for n: u64 in primes.into_iter() {
            assert!(is_prime(n));
        }
    }

    #[test]
    fn test_is_not_prime() {
        let not_primes: [u64; 6] = [0, 4, 6, 8, 9, 10];

        for n: u64 in not_primes.into_iter() {
            assert!(!is_prime(n));
        }
    }
}
```

corrode

MOVE TESTS TO DOC TESTS

src/prime.rs › mod tests

```rust
/// Check if a number is prime
///
/// # Examples
/// ```
/// use testing::prime::is_prime;
/// assert!(!is_prime(0));
/// assert!(is_prime(2));
/// assert!(is_prime(3));
/// assert!(!is_prime(4));
/// ```
pub fn is_prime(n: u64) -> bool {
    if n < 2 {
        return false;
    }
    for i: u64 in 2..n {
        if n % i == 0 {
            return false;
        }
    }
    true
}
```

corrode

MOVE TESTS TO DOC TESTS

testing
0.1.0

In
testing::prime

Functions

is_prime

Click or press 'S' to search, '?' for more options...

Function testing::prime::is_prime

source · [-]

```
pub fn is_prime(n: u64) -> bool
```

[-] Check if a number is prime

Examples

```
use testing::prime::is_prime;
assert!(!is_prime(0));
assert!(is_prime(2));
assert!(is_prime(3));
assert!(!is_prime(4));
```

corrode

HIDE LINES IN DOC TEST

src/prime.rs › mod tests

```rust
/// # Examples
/// ```
/// # use testing::prime::is_prime;
/// assert!(!is_prime(0));
/// assert!(is_prime(2));
/// assert!(is_prime(3));
/// assert!(!is_prime(4));
/// ```
pub fn is_prime(n: u64) -> bool {
    if n < 2 {
        return false;
    }
    for i: u64 in 2..n {
        if n % i == 0 {
            return false;
        }
    }
    true
}
```

corrode

MOVE TESTS TO DOC TESTS

testing
0.1.0

In
testing::prime

Functions

is_prime

Click or press 'S' to search, '?' for more options...

Function testing::prime::is_prime

source · [-]

```
pub fn is_prime(n: u64) -> bool
```

[-] Check if a number is prime

Examples

```
assert!(!is_prime(0));
assert!(is_prime(2));
assert!(is_prime(3));
assert!(!is_prime(4));
```

corrode

TEST DOUBLES

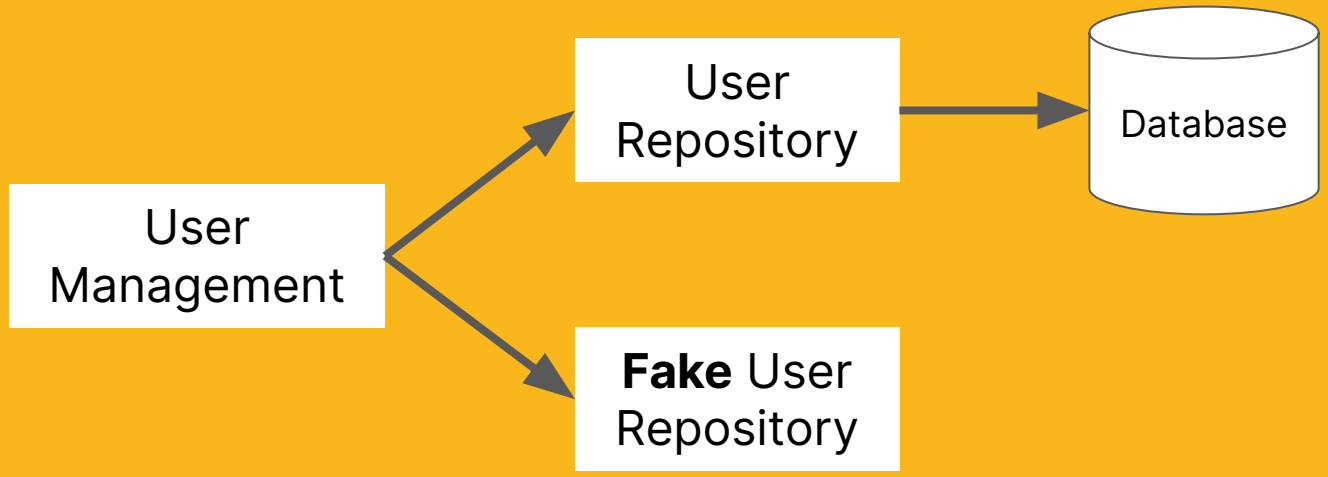
EXAMPLE

YOU CAN USE TRAITS FOR TEST DOUBLES

WHAT ARE TRAITS?

- Traits are like interfaces
- A struct can implement multiple traits

EXAMPLE



corrode

EXAMPLE

```rust
use crate::Result;

/// Repository trait.
///
/// This trait defines the basic operations that a repository should implement.
pub trait Repository<T> {
    /// Get all entities from the repository.
    fn find_all(&self) -> Result<Vec<T>>;
    /// Find an entity by its id.
    fn find_by_id(&self, id: i32) -> Result<Option<T>>;
}
```

corrode

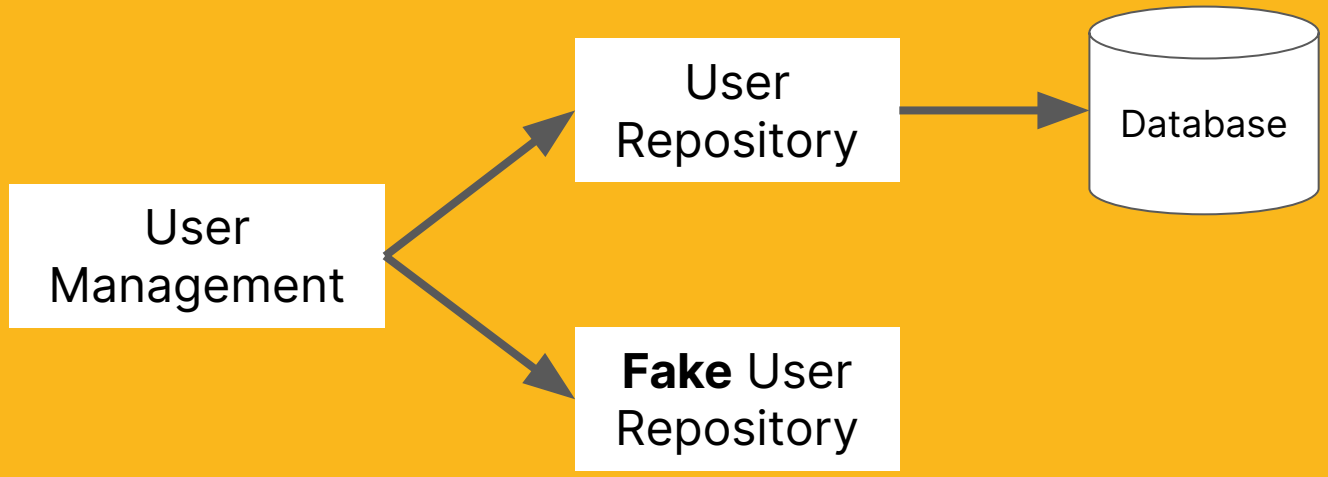
EXAMPLE

```
16 > impl Repository<User> for UserRepository {···
53   }
```

```
  #[cfg(test)]
> impl Repository<User> for FakeUserRepository {···
  }
```

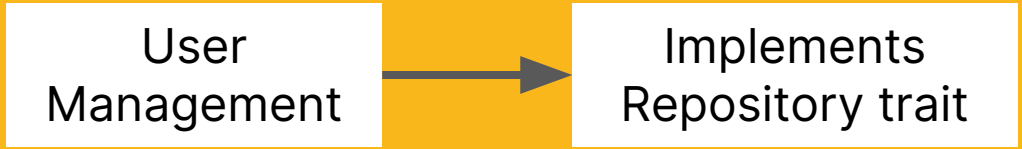
corrode

EXAMPLE



corrode

EXAMPLE



EXAMPLE

SEE DETAILS IN SAMPLE REPOSITORY

TIPS AND TRICKS

TIP 1: YOU CAN RETURN RESULT FROM TESTS

```rust
#[test]
fn some_function() -> Result<(), String> {
    assert_eq!(some_fallible_function("foo")?, 1);
    Ok(())
}
```

corrode

TIP 2: YOU CAN IGNORE FAILING TESTS

```
#[test]
fn some_test_which_currently_fails() {
    todo!()
}
```

corrode

TIP 2: YOU CAN IGNORE FAILING TESTS

```
#[test]
#[ignore]
fn some_test_which_currently_fails() {
    todo!()
}
```

corrode

TIP 2: YOU CAN TEST FOR PANICS

```rust
#[test]
#[should_panic]
fn test_panic() {
    panic!("This test should panic");
}
```

corrode

TIP 3: `#[cfg(test)]` IS SUPER HELPFUL

```rust
struct Foo {
    x: i32,
}

// Normal implementation of Foo
impl Foo {
    // ...
}
```

corrode

TIP 3: `#[cfg(test)]` IS SUPER HELPFUL

```rust
// Additional functionality for Foo only available in tests
#[cfg(test)]
impl Foo {
    fn random_number() -> i32 {
        42
    }

    fn special_foo_constructor() -> Foo {
        Foo { x: 42 }
    }
}
```

corrode

TIP 3: `#[cfg(test)]` IS SUPER HELPFUL

```rust
#[test]
fn test_foo() {
    assert_eq!(Foo::random_number(), 42);

    let foo: Foo = Foo::special_foo_constructor();
    assert_eq!(foo.x, 42);
}
```

corrode

TEST ISOLATION

TYPICAL THINGS TO ISOLATE

- File system
- Network
- `stdio`/`stderr`

TYPICAL THINGS TO ISOLATE

1. File system → `tempdir` crate
2. Network → `mockito` crate
3. `stdio`/`stderr` → `assert_cmd` crate

WHY IS USING THE FILE SYSTEM IN TESTS A BAD IDEA?

- Race conditions between tests
- Not reproducible
- Cleanup could be forgotten
- Differences between file systems
- Security issues

USE THE TEMPFILE CRATE INSTEAD

```rust
use tempfile::tempfile;
use std::io::{self, Write};

// Create a file inside of `std::env::temp_dir()`.
let mut file = tempfile()?;

writeln!(file, "Brian was here. Briefly.")?;
```

corrode

WHY SHOULDN'T YOU DEPEND ON THE NETWORK IN YOUR TESTS?

- Servers can be down
- Slow tests
- Servers can rate limit you
- Servers can block you
- Overloading the network
- Flaky tests are the result
- Hard to test edge-cases and error conditions

```rust
#[cfg(test)]
mod tests {
  #[test]
  fn test_something() {
    // Request a new server from the pool
    let mut server = mockito::Server::new();

    // Use one of these addresses to configure your client
    let host = server.host_with_port();
    let url = server.url();

    // Create a mock
    let mock = server.mock("GET", "/hello")
      .with_status(201)
      .with_header("content-type", "text/plain")
      .with_header("x-api-key", "1234")
      .with_body("world")
      .create();

    // Any calls to GET /hello beyond this line will respond with 201, the
    // `content-type: text/plain` header and the body "world".

    // You can use `Mock::assert` to verify that your mock was called
    // mock.assert();
  }
}
```

corrode

WHY SHOULDN'T YOU DEPEND ON `stdio` IN YOUR TESTS?

- `stdio` is often used for command-line tools
- Writing tests for CLI tools is quite tough to do manually

USE THE ASSERT_CMD CRATE INSTEAD

```rust
use assert_cmd::Command;

let mut cmd = Command::cargo_bin("bin_fixture").unwrap();
let assert = cmd
    .arg("-A")
    .env("stdout", "hello")
    .env("exit", "42")
    .write_stdin("42")
    .assert();
assert
    .failure()
    .code(42)
    .stdout("hello\n");
```

corrode

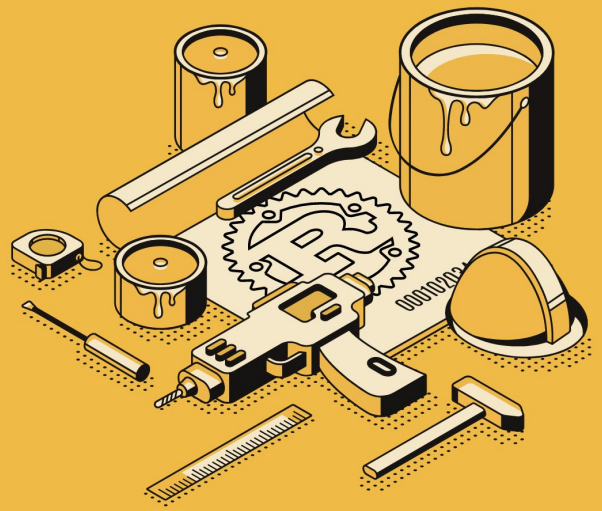
SPEEDING UP TESTS

TIPS FOR SPEEDING UP TESTS

- Unit tests > integration tests
- Use a single compilation unit for integration tests
- Try `cargo nextest` for ~60% faster text execution

RUST INSIGHTS

TIPS FOR FASTER RUST COMPILE TIMES



https://corrode.dev/blog/tips-for-faster-rust-compile-times/#test-execution

corrode

THE RUST TESTING ECOSYSTEM

“TABLE-BASED” TESTING

- `rstest`
- `test-case`

RSTEST EXAMPLE

```rust
#[test]
fn test_fibonacci() {
    assert_eq!(fibonacci(0), 0);
    assert_eq!(fibonacci(1), 1);
    assert_eq!(fibonacci(2), 1);
    assert_eq!(fibonacci(3), 2);
    assert_eq!(fibonacci(4), 3);
    assert_eq!(fibonacci(5), 5);
    assert_eq!(fibonacci(6), 8);
    assert_eq!(fibonacci(7), 13);
    assert_eq!(fibonacci(8), 21);
    assert_eq!(fibonacci(9), 34);
    assert_eq!(fibonacci(10), 55);
}
```

```rust
#[rstest]
#[case(0, 0)]
#[case(1, 1)]
#[case(2, 1)]
#[case(3, 2)]
#[case(4, 3)]
#[case(5, 5)]
#[case(6, 8)]
#[case(7, 13)]
#[case(8, 21)]
#[case(9, 34)]
#[case(10, 55)]
fn fibonacci_test(#[case] input: u64, #[case] expected: u64) {
    assert_eq!(expected, fibonacci(input))
}
```

RSTEST EXAMPLE

```rust
#[rstest]
#[case(0, 0)]
#[case(1, 1)]
#[case(2, 1)]
#[case(3, 2)]
#[case(4, 4)]
#[case(5, 5)]
#[case(6, 8)]
#[case(7, 13)]
#[case(8, 21)]
#[case(9, 34)]
#[case(10, 55)]
fn fibonacci_test(#[case] input: u64, #[case] expected: u64) {
    assert_eq!(expected, fibonacci(input))
}
```

corrode

RSTEST EXAMPLE

```
---- exercises::test_tables::tests::fibonacci_test::case_05 stdout ----
thread 'exercises::test_tables::tests::fibonacci_test::case_05' panicked at src/exercises/test_tables.rs:19:9:
assertion `left == right` failed
  left: 4
 right: 3
```

corrode

EXPLORATORY TESTING

RUST TESTING ECOSYSTEM

- `quickcheck`
- `proptest`

pretty_assertions

```
thread 'main' panicked at 'assertion failed: `(left == right)`

Diff < left / right > :
 Some(
     Foo {
<        lorem: "Hello World!",
>        lorem: "Hello Wrold!",
         ipsum: 42,
         dolor: Ok(
<            "hey",
>            "hey ho!",
         ),
     },
 )

', examples/standard_assertion.rs:20:5
```

corrode

QUESTIONS?

SURVEY: EXPECTATIONS MET?

BREAK

corrode

WELCOME BACK!
LET’S DO SOME PRACTICAL EXERCISES

corrode

FINAL NOTES

ADVANCED TESTING STRATEGIES

- The following architectures work great with testing
  - Hexagonal architecture
  - Onion Architecture
  - Ports and Services

Read more at https://alexis-lozano.com/hexagonal-architecture-in-rust-1/

TEST COVERAGE

```
cargo tarpaulin --out Html
```

BENCHMARK TESTS

- Criterion
- Divan

THANK YOU

corrode.dev

corrode

OTHER WORKSHOPS

corrode

LEARNING RUST BY COUNTING WORDS

Voraussetzungen: Grundlegende Programmierkenntnisse

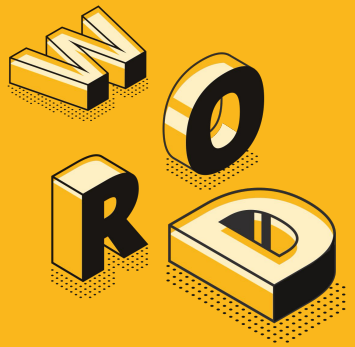
Dauer: 8 Stunden, verteilt auf zwei Sitzungen à 4 Stunden

Lernumgebung: Die Teilnahme erfolgt in der Regel remote, für größere Gruppen auch vor Ort.

Kursmaterial: Präsentation, Dokumentation, Codebeispiele, Live-Betreuung

Lernziele: Rust-Grundlagen, cargo, Fehlerbehandlung, Tests

Sprache: Englisch oder Deutsch



corrode

WRITE YOURSELF A SHELL IN RUST

Voraussetzungen: Grundlegende Rust-Kenntnisse

Dauer: 8 Stunden, verteilt auf zwei Sitzungen à 4 Stunden

Lernumgebung: Die Teilnahme erfolgt in der Regel remote, für größere Gruppen auch vor Ort.

Kursmaterial: Präsentation, Dokumentation, Codebeispiele, Live-Betreuung

Lernziele: Idiomatischer Code, Testing, Design-Patterns, Erweiterte Konzepte (Lifetimes, Parsing, Prozesskontrolle)

Sprache: Englisch oder Deutsch

corrode

Rust in Production Podcast

corrode.dev/podcast

corrode

SEASON 02

Thunderbird with Brendan Abolivier	2024-05-30 S02 E03
AMP with Carter Schultz	2024-05-16 S02 E02
curl with Daniel Stenberg	2024-05-02 S02 E01

SEASON 01

Season Finale	2024-03-07 S01 E07
Sentry with Arpad Borsos	2024-02-22 S01 E06
Tweede Golf with Folkert de Vries	2024-02-08 S01 E05
Arroyo with Micah Wylde	2024-01-25 S01 E04
Apollo with Nicolas Moutschen	2024-01-11 S01 E03
PubNub with Stephen Blum	2023-12-28 S01 E02
InfluxData with Paul Dix	2023-12-14 S01 E01
Teaser	2023-12-11 S01 E00

corrode

Rust in Production Podcast

corrode.dev/podcast

corrode