ZEISS Digital Innovation

Our experience in figures

Berlin
Leipzig
Görlitz
Dresden
München
Miskolc
Budapest

7

Locations

in Dresden, Munich, Berlin, Leipzig, Görlitz, Miskolc and Budapest (Hungary)

630

Permanent employees

in Germany and Hungary

30+

Years of experience in IT & software development

ISO 9001 & ISO 27001 certified

2

Business Lines

Health & Life Science Solutions / Manufacturing Solutions

4x

Leader (Agile) Software Development

Award by an international benchmark for analytics in 2022, 2021, 2018, 2016

ZEISS Digital Innovation

Accelerate innovation through strategic synergies

We enable innovative digital solutions for our clients inside and outside ZEISS

Medical
Technology

Industrial
Quality & Research

Semiconductor
Manufacturing
Technology

Consumer
Markets

Focus industries

**Healthcare /
Life Science**

**Industry /
Manufacturing**

Rust in Testing

Maria Shalnova-Weinzierl
ZEISS Digital Innovation

Agenda

01 Introduction – Rust in a Nutshell

02 Set up a Test Environment

03 Unit Tests with Rust

04 Mock API - API-Tests with Rust

05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

1 | Introduction
Testing in Rust

01 Introduction – Rust in a Nutshell

02 Set up a Test Environment

03 Unit Tests with Rust

04 Mock API - API-Tests with Rust

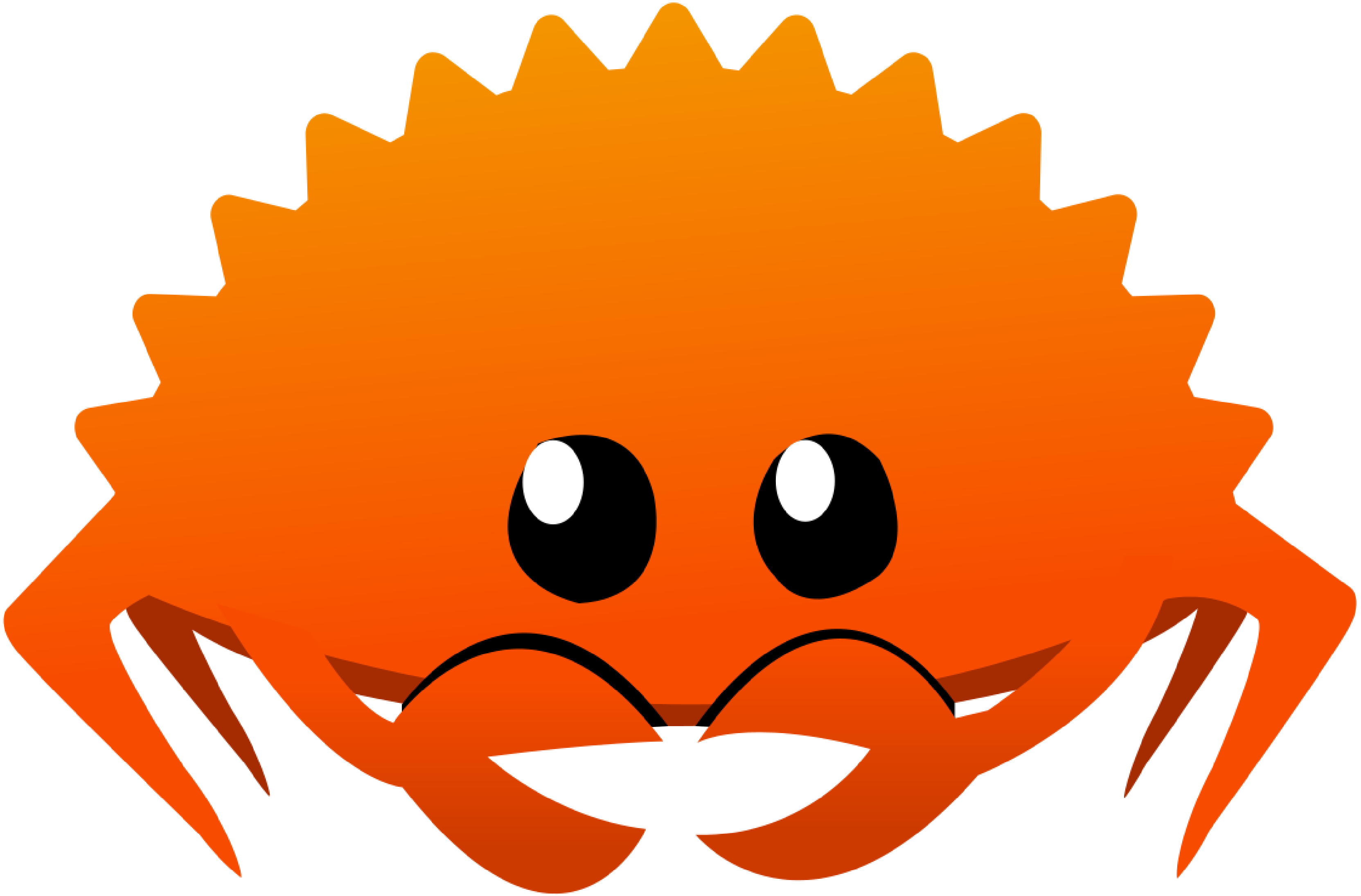
05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

1 | Introduction
Testing in Rust

- Graydon Hoare
- Rust Foundation
- Main Advantages of Rust



1 | Introduction

Testing in Rust

- Type Safety
- Concept of Ownership
- Features of Object Oriented Programming
- Influenced by Scripting Languages and Functional Programming

1 | Rust in a Nutshell

Main Concepts

- Borrowing and References
- Lifetimes: Ensuring Valid References
- The **match** Expression

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-NC-ND

1 | Rust in a Nutshell

Main Concepts

- Working with Slices, Arrays, and Vectors
- Iterators and Closures
- Concurrency with Threads and Channels

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-NC-ND

1 | Rust in a Nutshell

cargo

- package manager
- Install Cargo
- cargo new
- cargo test

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-NC-ND

2 | Rust in a Nutshell

- cargo
- cargo and rustc
- Compiler Error Index

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-NC-ND

1 | Rust in a Nutshell

package layout

```
├── Cargo.lock
├── Cargo.toml
├── src/
│   ├── lib.rs
│   ├── main.rs
│   └── bin/
│       └── multi-file-executable/
│           ├── main.rs
│           └── some_module.rs
└── tests/
    ├── some-integration-tests.rs
    └── multi-file-test/
        ├── main.rs
        └── test_module.rs
```

2 | Set Up a Test Environment

Testing in Rust

01 Introduction – Rust in a Nutshell

02 Set Up a Test Environment

03 Unit Tests with Rust

04 Mock API - API-Tests with Rust

05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

2 | Set Up a Test Environment

1. **Install Rust**: https://www.rust-lang.org/tools/install

2. **Check Installation:** `rustc --version`

3. **cargo new my_project**

4. **cd my_project**

2 | Set Up a Test Environment

1. **cargo build**

2. **cargo run**

3. **cargo test**

4. **cargo doc --open**

2 | Set Up a Test Environment

```
> cargo new test_env

> cargo run
```

Cargo.toml

```toml
[package]
name = "test_env"
version = "0.1.0"
edition = "2021"

[dependencies]
```

2 | Set Up a Test Environment

Demo

Test Environment is Ready

Exercise 0

1) Set Up Test Environment
2) Test the Set Up with cargo test

3 | Unit Tests with Rust

Testing in Rust

01 Introduction – Rust in a Nutshell

02 Set Up a Test Environment

03 Unit Tests with Rust

04 Mock API - API-Tests with Rust

05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

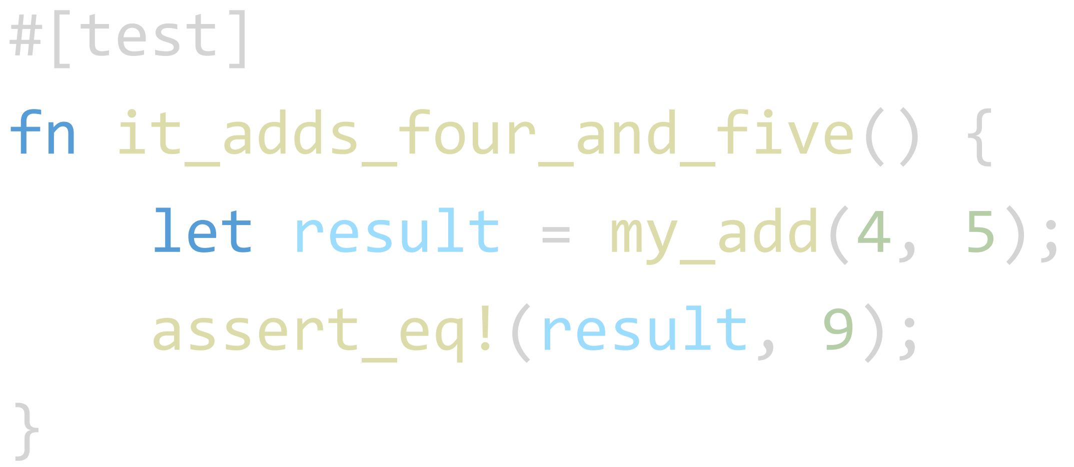
3 | Unit Tests with Rust

> “Tests are Rust functions that verify that the non-test code is functioning in the expected manner. The bodies of test functions typically perform some setup, run the code we want to test, then assert whether the results are what we expect.”

https://doc.rust-lang.org/rust-by-example/testing/unit_testing.html

3 | Unit Tests with Rust

```
> cargo test
   Compiling calculation_test v0.1.0
(C:\projects\rust2025_tests\rust2025\rust
2025\exercise1\calculation_test)
    Finished `test` profile [unoptimized +
debuginfo] target(s) in 0.69s
     Running unittests src\main.rs
(target\debug\deps\calculation_test-
1ca22128ac602ad0.exe)
running 1 test
test tests::it_adds_four_and_five ... ok
```



```rust
#[test]
fn it_adds_four_and_five() {
    let result = my_add(4, 5);
    assert_eq!(result, 9);
}
```

3 | Unit Tests with Rust

Demo

Unit Tests are Running

Exercise 1

1. According to the given pattern, develop unit tests for the other three arithmetic operations

2. Use **cargo test** to run the new developed tests

Exercise 1

1. According to the given pattern, develop negative unit tests for the other three arithmetic operations

2. Use **cargo test** to run the new developed tests

3 | Unit Tests with Rust

Using googleTest:

Add a new dependency in Cargo.toml

4 | Mock API – API Tests with Rust

Testing in Rust

01 Introduction – Rust in a Nutshell

02 Set Up a Test Environment

03 Unit Tests with Rust

04 Mock API – API-Tests with Rust

05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

4 | Mock API - API-Tests with Rust

WireMock

```
cargo add wiremock --dev
```

4 | Mock API - API-Tests with Rust

```
> cargo test
running 1 test
test test ... Ok

test result: ok. 1 passed; 0 failed;
0 ignored; 0 measured; 0 filtered
out; finished in 0.01s
```

```
let status =
reqwest::get(format!("{}/missing
", &mock_server.uri()))
        .await
        .unwrap()
        .status();
 assert_eq!(status.as_u16(),
404);
```

4 | Mock API - API-Tests with Rust

Demo

API Tests are running

4 | Mock API - API-Tests with Rust

https://github.com/msh707/workshop2025_wiremock

4 | Mock API - API-Tests with Rust - Exercise

Exercise 2

1) Develop the tests for the HTTP response status codes 403, 400, 201 and 401
2) Run the tests using **cargo run**

5 | Integration Tests and Snapshot Tests

Testing in Rust

01 Introduction – Rust in a Nutshell

02 Set up a Test Environment

03 Unit Tests with Rust

04 Mock API – API-Tests with Rust

05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

5 | Integration Tests in Rust

Unit tests are testing one module in isolation at a time: they're small and can test private code. **Integration tests** are external to your crate and use only its public interface in the same way any other code would. Their purpose is to test that many parts of your library work correctly together.

https://doc.rust-lang.org/rust-by-example/testing/integration_testing.html

5 | Integration Tests in Rust

```
> cargo test

running 1 test
test it_check_function ... ok

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

```rust
use integration_test::calculator::addition;

#[test]
fn it_check_function() {
    let result = addition(4, 1);
    assert_eq!(result, 5);
}
```

5 | Integration Tests in Rust

Negativ Test – flag #[should_panic]

```rust
#[test]
#[should_panic]
fn it_check_function_failed() {
    let result = addition(4, 1);
    assert_eq!(result, 500);
}
```

5 | Integration Tests in Rust

Demo

Integration Tests are Running

Exercise 4

1) According to the given pattern, develop integration tests for the other three arithmetic operations
2) Run the test using **cargo run**

5 | Snapshot Tests in Rust

- Snapshot Tests
- Crate Insta
- Test Structure
- Assertion Macros (the use of **serde::Serialize** is required)

5 | Snapshot Tests in Rust

- fundamentally different from unit and functional test
- comparing the current characteristics of an application with recorded expected values
- tests can be developed much faster

5 | Snapshot Tests in Rust – Crate Insta

- cargo add --dev insta --features yaml

- cargo insta test

- cargo insta review

5 | Snapshot Tests in Rust – Test Structure

```rust
#[test]
fn test_division() {
    let div = division(14, 2);
    insta::assert_yaml_snapshot!(div, @"");
}
```

5 | Snapshot Tests in Rust – Assertion Macros

Macro	Usage
assert_csv_snapshot!	for comparing CSV serialized output. (requires the csv feature)
assert_toml_snapshot!	for comparing TOML serialized output. (requires the toml feature)
assert_yaml_snapshot!	for comparing YAML serialized output. (requires the yaml feature)
assert_ron_snapshot!	for comparing RON serialized output. (requires the ron feature)
assert_json_snapshot!	for comparing JSON serialized output. (requires the json feature)
assert_compact_json_snapshot!	for comparing JSON serialized output while preferring single-line formatting. (requires the json feature)

5 | Snapshot Tests in Rust - Demo

Demo

Developing snapshot tests

Running snapshot tests

Review of snapshot tests

6 | Documentation Tests, Benchmarking and Code Coverage

Testing in Rust

01 Introduction – Rust in a Nutshell

02 Set up a Test Environment

03 Unit Tests with Rust

04 Mock API - API-Tests with Rust

05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

6 | Documentation Tests in Rust

- HTML documentation
- Documentation is based on comments in code
- Two slashes with an exclamation mark in the documentation header
- Comments with three slashes in the begin of the line
- cargo doc

6 | Documentation Tests in Rust

Demo

Creation of Documentation

6 | Benchmarking

- crate criterion
- define benchmarks
- cargo bench

6 | Benchmarking - Demo

Demo

Creation of benchmark report

6 | Code Coverage

- cargo install cargo-tarpolin
- cargo tarpaulin
- cargo tarpaulin –out html

6 | Code Coverage

Demo

Creation of code coverage report

7 | Conclusion
Testing in Rust

01 Introduction – Rust in a Nutshell

02 Set up a Test Environment

03 Unit Tests with Rust

04 Mock API - API-Tests with Rust

05 Integration Tests and Snapshot Tests

06 Documentation Tests, Benchmarking and Code Coverage

07 Conclusion

7 | Conclusion

1. **From a side project to the world-renowned programming language**

2. **Improved Concepts**

3. **Many useful built-in features**

4. **A lot of fun**

References

- https://www.rust-lang.org/
- https://doc.rust-lang.org/
- https://bootcamp.cvn.columbia.edu/blog/new-programming-languages/
- https://en.wikipedia.org/wiki/Rust_(programming_language)
- https://commonmark.org/
- https://docs.rs/insta/latest/insta/
- https://docs.rs/criterion/latest/criterion/
- https://rustfoundation.org/
- https://github.com/msh707/workshop_rust_tests_solutions

Your Feedback

https://forms.office.com/e/HEPSvC4HGq?origin=lprLink

ZEISS

Seeing beyond