1 Einleitung

Der Anfang von Rust wurde mit Hilfe von einem persönlichen Hobby-Projekt des Mozilla-Mitarbeiters Graydon Hoare für die Erschaffung einer Browser-Engine für Firebox realisiert[2].Ein Jahr später hat dieses Projekt das Interesse von Mozilla geweckt und so begann das Sponsoring von Rust[2]. Später wurde die Sprache als Open-Source bezeichnet und im Bezug darauf hat sie eine große finanzielle Unterstützung nicht nur von Mozilla,sondern auch von ihren aktiven Community gekriegt[1][2].Im 2012 wurde die erste Alpha Version von Rust veröffentlicht und am 8. Februar 2021 wurde die Gründung der Rust Foundation von den fünf Gründungsunternehmen (AWS, Huawei, Google, Microsoft und Mozilla) offiziell bekannt gegeben[11]. Rust kann als ein Low-level statisch-geschriebene multi-paradigma Programmiersprache beschrieben werden, welche sich auf Sichterheit und Leistung konzentriert. Sie wurde mit dem Ziel entwickelt Probleme zu lösen, mit dennen sich C und C++ für lange Zeit beschäftigt haben, wie z.B Speicherfehler und das Bauen von nebenläufigen Programmen.

2 Technische Grundlagen

Die Programmiersprache benutzt einen Compiler wie C++ und kein Interpreter wie Python oder Javascript.

Heutzutage steht immer im Fokus die Debatte, ob Rust eine objektorientierte Programmiersprache ist.Die vier wichtigsten Säulen von OOP sind Verkapselung, Polymorphism, Abstraktion und Inheritance. Rust beinhaltet drei von den vier Eigenschaften[10]:

1. Verkapselung ist sozusagen die Umhüllung von Daten und Informationen in einer Einheit, die in C++ Klasse gennant wird. In Rust gibt es keine Klassen-Implementation. Dieselbe Funktionalität wird aber in Rust durch Verwendung von Strukturen erreicht[10].

2. Polymorphism heißt mehrere Formen oder unterschiedliche Implementationen einer Funktion. Es gibt zwei Typen von Polymorphism: Kompilierzeit- und Laufzeit-Polymorphism. Kompilierzeit-Polymorphism kann durch Operator oder Methodenüberladung und bzw. Laufzeit-Polymorphism durch Methodenüberladung in C++ erreicht werden. In der Programmiersprache Rust kann Polymorphism nur durch Trait erreicht werden[10].

3. Abstraktion ermöglicht es nur die wichtigsten Informationen zu zeigen und andere Details zu verbergen, indem Traits verwendet werden und so können die Methoden ohne deren Implementationen definiert werden[10].

4.Mit Hilfe von Inheritance kann man der Quellcode Wiederverwenden und gleichzeitig sicherer machen. Im Bezug auf die Wiederverwendbarkeit kann man die Eigenschaften einer Elternklasse in anderen Kinderklassen einsetzen. Trotzdem ist der Einzatz dieser Methode heutzutage nicht so aktuell wegen der niedrigeren Sicherheit. Der Grund dafür ist die gemeinsame Information zwischen den Haupt- und den Subklassen. Auf diese Weise wird der Kode unflexibel sein.Diesbezüglich unterstützt Rust keine Inheritance[10].

Obwohl Rust sehr ähnlich der Programmiersprache C ist, wurde das Design von Rust von vielen vorhandenen Sprachen und Techniken inspiriert. Ein wesentlicher Einfluss darauf hat die funktionale Programmierung.Rust wird nicht von allen Experten einstimmig als funktionale Programmiersprache klassifiziert. Ein Grund dafür ist, dass diese Programmiersprache im Vergleich zu den anderen funktionalen Programmiersprachen veränderlichen Zustände anbietet.Während sich die funktionale Programmiersprachen darauf konzentrieren, einen veränderlichen Zustand zu vermeiden, versucht Rust nur ein Teil der Gefahr zu verhindern.Andere Gründe sind der Mangel an funktionalen Datastrukturen wie z.B Tail-Call Optimierung, die die funktionale Programmierung ermöglichen[3].

Rust ist auch ein statisch typisierte Sprache.Das bedeutet, dass während der Kompilierungszeit alle Typen bekannt werden. Das Management der Komplexität ist einer der wichtigsten Aspekten bei der Programierung.Deswegen spielen die statisch typisierte Sprachen eine wichtige Rolle, indem sie Überblick über die Vorgänge im Code behalten. Im Bezug darauf ist es schwieriger falsche Programmen mit Rust zu schreiben. In dieser Sprache braucht man auch nicht die Typen der Variable mehrmals zu wiederholen, um eine langfristige Wartbarkeit zu fördern[4].

Die Programmiersprache kann nicht nur als High-Level-, sondern auch als Low-Level-Programmiersprachen bezeichnet werden.Ähnlich wie C/C++ ist Rust sehr nah an der Hardware, was für die höhe Geschwindigkeit sorgt.Trotzdem kann man mit Rust relativ einfach programmieren, was man es sonst von High-Level-Sprachen

kennt[6].

Eine der Eingeschften von Rust ist, dass man ohne Probleme eine gemainsame Lösung erstellen kann, die auf verschieden Betriebssystemen Linux, MacOS, Windows und anderen Plattformen funktionieren wird (Cross-Platform ist unterstützt)[4].

3 Architektur

Im Bezug auf dem Syntaxis ähnelt auf den ersten Blick sehr stark C oder C++ , wo man auch mit Funktionen, Schleifen, Abfragen, Konstanten und Variablen arbeitet.Trotzdem besitzt Rust auch eigene Eigenschaften im Vergleich zu den anderen Sprachen[6]wie z.B die Definition neuer Funktionen durch den Befehl „fn“ oder die Arbeit der Sprache mit Makros, die sich durch ein Ausrufezeichen am Ende des Begriffs auszeichnen[6].

Syntax-Beispiele :

Bsp1:

```
 fn main() {
    println!("Hello, World!");
}
```

Von dem Beispiel ist zu sehen dass Funktionendefinitionen mit “fn” anfangen und dass die print Funktion im Rust “println!” heißt[1].

Variablen und Konstanten

In Rust das Schlüsselwort ‚let‘ verwendet, um eine Variable zu deklarieren. Eine bestehende Variable kann in Rust erneut deklariert werden und „überschattet“ dann die bestehende Variable. Im Unterschied zu vielen anderen Sprachen kann der Wert einer Variable nicht ohne weiteres geändert werden[1]: **Bsp 2**:

```
// Variable ‚alter‘ deklarieren und Wert auf ‚42‘ setzen
let alter = 42;
// Wert der Variable ‚alter‘ kann nicht verändert werden
alter = 49; // Kompilier-Fehler
// mit erneutem ‚let‘ kann die Variable überschrieben werden
let alter = 49;
```

Um den Wert einer Variable als im Nachhinein änderbar zu markieren, bringt Rust das ‚mut‘-Schlüsselwort mit. Der Wert einer mit ‚mut‘ deklarierten Variablen lässt sich verändern[1].

Bsp 3:

```
let mut gewicht = 78;
gewicht = 75;
```

Die wiederholte Ausführung eines Code-Blocks mittels Schleifen ist auch als „Iteration“ bekannt. Oft wird über den Elementen eines Containers iteriert. Wie Python kennt Rust das Konzept des „Iterators“. Ein Iterator abstrahiert den sukzessiven Zugriff auf die Elemente eines Containers. Schauen wir uns ein Beispiel an[1].

Bsp 4:

```
// Liste mit Namen
let namen = ["Jim", "Jack", "John"];
// ‚for‘-Schleife mit Iterator in die Liste
for name in namen.iter() {
    println!("Hallo, {}", name);

}
```

Auch die Speicherverwaltung gilt als ausgesprochen sicher, indem die Sprache keine Nullzeiger und Data-races erlaubt. Im Vergleich zu anderen Programmiersprachen verwendet man bei dem Zeiger weder den Wert “valid” noch “NULL”. Trotzdem bietet die Rust einen Optionstyp, mit dessen Hilfe herausgefunden werden kann, ob ein Zeiger entweder den Wert “Some” oder “None” besitzt. Die Sprache stellt auch zur Verfügung das Schlüsselwort “unsave”,die genutzt wird, um ein unsicherer Code zu schreiben[12]. Eine der Eigenschaften von Rust ist ,dass die Sprache als “memory safe” bezeichnet ohne die Verwendung von “garbage collector” wird,indem beim Komilieren eine Übreprüfung durchgeführt wird, ob es Feher im Speicher gibt[6].Seit vielen Jahren ist der Speicher einer der beliebigsten Angriffspunkten von Hacker.Der Grund dafür ist, dass beim Volllaufen der Speicher liefert einen Fahler bzw. eine Schwäche im System, die ausgenutzt werden kann. Die Aufgabe eines “garbage collector” ist die unnötige Objekte aus dem Speicher wegzunehmen, was die Ausführung des Codes verlangsamt[6].

Das Ownerschip Modell ist das einzigartigsten Merkmale von Rust[9], indem ein bestimmter Wert nur zu einer Variablen gehören kann[6].Das Besitztum ist eng verknüpft mit dem Wert von Variablen und deren „Lifetime“ und mit der Speicherverwaltung von Objekten auf dem „Heap“-Speicher. Wenn eine Variable den Gültigkeitsbereich („Scope“) verlässt, wird ihr Wert zerstört und so wird bzw. der Speicher freigegeben. Im folgende Beipiel dieses Konzept demonstriert[1].

Bsp 5:

```
    let name = String::from("Peter Mustermann");
let _name = name;
println!("{}, world!", name);
// Kompilier-Fehler, da der Wert von ‚name‘ an ‚_name‘ weitergereicht wurde[1]
```

Der Ownership im Rust wird mit Hilfe von einer der wichtigsten Komponente der Sprache realiesiert:der Borrow-Checker, dessen Aufgabe ist herauszufinden, welche Daten an welchen Stellen verwendet werden. Diesbezüglich kann den Borrow-Checker bestimmen welche Daten freigegeben oder intialisiert werden müssen.Auf diese Weise kriegt man die Geschwindigkeit und die Effizienz einer manuellen Verwaltung.[9].Mit der Hilfe des Borrow-Checkers können auch die “concurrencys” vereinfacht werden, indem “Data-races” während der Kompilierzeit vermieden werden.Data-races treten auf, wenn zwei Threads gleichzeitig auf denselben Speicher zugreifen wollen, was zu unangenehmenen Konsequenzen führen kann[3].

Das Typsystem in Rust unterstützt ein Mechanismus ähnlich den Typklassen von Haskell, die als Traits bezeichnet werden.Ein Trait in Rust ist eine Gruppe von Methoden, die für einen unbekannten Typ definiert sind: “Self”. Die können aber für jeden Datentyp implementiert werden[15]. Traits sind eine abstrakte Definition des gemeinsamen Verhaltens zwischen verschiedenen Typen. In gewisser Weise sind Traits zu Rust, was Interfaces zu Java oder abstrakte Klassen zu C ++ sind[14]. Eine Traitsmethode kann auf andere Methoden innerhalb dieses Traits zugreifen. In Rust erfüllen Implementierungen eine ähnliche Rolle wie Klassen in anderen Sprachen,wo man Methoden definieren kann. Implementierungen werden mit dem Schlüsselwort “impl” definiert.Polymorphismus in dieser Programmiersprache kann durch Traits bereitgestellt werden. Die Traits ermöglichen nich nur die Definition, sondern auch die Mischung der Methoden in einer Implementierung(impl).Trotzdem werden die Feldern nicht mit Implementierung oder Traits, sondern mit Strukturen definiert[16]. Mit der Hilfe von diesen Eigenschaften kann das Diamantproblem der Vererbung wie in C ++ verhindert werden, bei dem die Kindernklassen Merkmale von mehr als einer Elternklasse erben können[10]. Eine klassische Vererbung mit Srukturen im Rust existiert nicht. Im Bezug darauf wird den Prinzip “composition over inheritance” mit Hilfe den Traits verwendet. Während der Vererbung eine Klasse von einer anderen ableitet, definiert die Komposition eine Klasse als die Summe ihrer Funktionalitäten[18]. Die Implementierung von “generics” ähnelt der typischen Implementierung im C ++.Die generische Funktion kann “type-checked” werden, sobald sie definiert ist. Dies steht im Gegensatz zu C ++ , wo in der Aufrufphase keine Typen überprüft werden. Stattdesen wird die “Template” solange erweitert, bis sie entweder erfolgreich ist oder eine Operation erreicht, die von diesem bestimmten Typ nicht unterstützt wird[13].

Im Bezug auf die Leistung ist Rust mit C++17 vergleichbar und unbedigt besser als Sprachen wie Python.Einige der Gründe dafür sind nicht nur der Mangel an einem “Garbage collector”, sondern auch die „zero cost abstractions“, die für Geschwindigkeiten während der Laufzeit sorgen. [4][6]. Auf diese Weise kann man sowohl abstrakt programmieren, ohne Leistungsverluste hinzunehmen. Ausßerdem besitzt Rust keine Laufzeitüberprüfung und deswegen wird der falsche Kode direkt eliminiert. Auf diese Weiße wird das System vom fehlerhaften Kode geschützt.[4]

Weil viele aufwändige Fehlermeldungen beim Rust konzipiert werden, ist das einer der Gründe warum diese Programmiersprache nicht nur für erfahrene Programmierer,sondern auch für Anfänger zugänglich ist.Rust unterscheidet zwei Arten von Fehlern:“recoverable” und “unrecoverable”. Für “recoverable” Fehlern wie z.B “file not found” wird das Problem für den Benutzer behoben. Nicht “unrecoverable” Fehler sind immer ein Ergebnis von “bugs”, z. B. der Versuch, auf einen Speicherort jenseits des Endes eines Arrays zuzugreifen.Die meisten Programmierspra-

chen machen keinen Unterschied zwischen den beiden Arten von Fehlern, die mit Hilfe von Exceptions behandelt werden.Im Gegensatz dazu besitzt Rust keine Exceptions.Stattdessen hat die Sprache den Typ Ergebnis <T, E> für behebbare Fehler und die “panic!” Makro, das die Ausführung stoppt, wenn das Programm auf einen nicht behebbaren Fehler stößt.Im Vergleich zu anderen Sprachen, die kryptische Errors ausgeben, liefert Rust sinnvolle und hilfreiche Hinweise, wie man die Fehler beheben kann[8].

Die zunehmende Popularität von Rust erfordert einen schnellen Wachstum des Rust-Ökosystem.Ein wichtiger Faktor in dem riesigen Ökosystem ist, dass es mehr als nur eine Sprache oder ein Compiler ist. Die Qualität der Softwareproduktion erfordert viel mehr Aspekte, die von dem Rust-Ökosystem alle als sehr wichtig bezeichnet werden. Ein Beispiel dafür ist das Befehlszeilentool “Cargo”, das von Rust-Programmierern beim Verwalten von Abhängigkeiten, Ausführen von Tests und Generieren von Dokumentation verwendet wird.Außerdem hat Rust eine Community-Site “crates.io”, wo man Information über alle Bibliotheken finden kann[4].Diesbezüglich werden viele Konferenzen wie RustConf,RustBelt und RustFest angeboten, die als sehr nützliche Quelle für den Rustentwicklers genutzt werden können.

4 Anwendungsbeispiel

Die Verwendung von Rust wäre eine gute Variante bei der Entwicklung von Applikationen, wenn man nach einer höheren Leistung strebt.Andere Beispiele für den Einsatz dieser Programiersprache sind nicht nur die Arbeit mit riesigen Datenmengen, sondern auch die bessere Zuweisung von Ressourcen in Threads. Wenn die Sicherheit im Fokus steht ist die Verwendung von Rust eine gute Idee.Alle diese Eingenschaften könnte Rust als ideal für die Entwicklung von Spielen, Betriebs- und Dateisystemen oder Browserkomponenten bezeichnet werden[4].Der Einsatz von Rust wächst auch in vielen weltberühmten Unternehmen. Einige Firmen wie Cloudfare und Microsoft verwenden die Sprache wegen der bereitgestellten Speichersicherheit. Andere Gründe für die Einführung von Rust in Unternehmen wie Discord,Amazon,Figma sind die größere Performance und die leichtere “concurrencys” im Vergleich zu Programmiersprchen wie C[5].

5 Zusammenfassung

Im Vergleich zu anderen Programmiersprachen biete Rust nicht nur mehrere Sicherheit, sondern auch größere Performance.Die Programmiersprache besitzt keine komplett neuen Inovationen, sondern nutzt als Bausteil was man beispielsweise aus C/C++ kennt, bietet aber interessante neue Eigenschaften.Ein Beispiel dafür ist,dass viele weltberühmte Unternehmen Rust bevorzugen, um die Speicherproblemen mit C zu vermeiden.Außerdem macht diese Sprache die Programmierung auf niedrigerer Ebene zugänglicher und ist für “concurency”(Paralilität) fantastisch. Besonders wenn man bereits mit anderen Sprachen vertraut ist, ist der Umstieg nicht schwierig.Diesbezüglich steigt der Einsatz von Rust in den weltbrühmten IT-Unternehmen wie Amazon,Facebook oder Microsoft immer an.

6 Quellenverzeichnis

[1]Digital guide IONOS; https://www.ionos.de/digitalguide/websites/web-entwicklung/rust-tutorial/, zuletzt Besucht:15.04.2021

[2]Internet Archive; https://web.archive.org/web/20160609195720/https://www.rust-lang.org/faq.html#project, zuletzt Besucht:15.04.2021

[3]Gints Dreimanis; Introduction to Rust;Serokell;https://serokell.io/blog/rust-guide#rust-vs.-c%2B%2B ,19.08.2020.;19.08.2020

[4]Tapan Patel;7 Reasons Why You Should Use Rust Programming For Your Next Project;https://simpleprogrammer.com/rust-programming-benefits/Simpleprogrammer-Website;18.11.2020

[5]Gints Dreimanis; 9 Companies That Use Rust in Production ;Serokell;https://serokell.io/blog/rust-companies , 18.11.2020.;18.11.2020

[6]Digital guide IONOS; https://www.ionos.de/digitalguide/websites/web-entwicklung/rust-programmiersprache-vor-

gestellt/#:~:text=Rust%20ist%20eine%20Sprache%20f%C3%BCr,hohe%20Geschwindigkeiten%20w%C3%A4hrend%20der%20Laufzeiten, zuletzt Besucht:15.04.2021

[7]rust-lang.org; https://static.rust-lang.org/doc/stable/std/ops/trait.Add.html, zuletzt Besucht:15.04.2021

[8]Steve Klabnik & Carol Nichols:The Rust Programming Language;https://doc.rust-lang.org/book/ch09-00-error-handling.html; 2018

[9]Thomas Heartman; Understanding the Rust borrow checker; LogRocket;https://blog.logrocket.com/introducing-the-rust-borrow-checker/;28.07.2020

[10]Rahul Bhati; OOP Features of RUST;Knoldus;https://blog.knoldus.com/oop-features-of-rust/; 08.01.2019

[11]The Rust Core Team: Announcing Rust 1.0; https://blog.rust-lang.org/2015/05/15/Rust-1.0.html , 15.05.2015.

[12]Neil Brown: A taste of Rust; https://lwn.net/Articles/547145/ , 17.04.2013.

[13]GitHub;https://gist.github.com/brendanzab/9220415?fbclid=IwAR1hJbVtUF2LWmZJZHjtxMr2KoGJRonjo-ElHtJ1yOd3UF__ZG68uVaN7Zds; zuletzt Besucht:16.04.2021

[14]Anusheh Zohair Mustafeez:What are traits in Rust? ; edicative ; https://www.educative.io/edpresso/what-are-traits-in-rust ; zuletzt Besucht:16.04.2021

[15]Learning Rust ; https://learning-rust.github.io/docs/b5.impls_and_traits.html ; zuletzt Besucht:16.04.2021

[16]Rust by example ; https://doc.rust-lang.org/rust-by-example/index.html;zuletzt Besucht:16.04.2021

[17]Inheritance with Traits ;https://riptutorial.com/rust/example/22917/inheritance-with-traits ; zuletzt Besucht:16.04.2021

[18] Rafael Del Nero: Inheritance versus composition,https://www.infoworld.com/article/3409071/java-challenger-7-debugging-java-inheritance.html; 08.01.2020