Interview mit Nicole Rauch

Funktionale Programmierung – ein Paradigmenwechsel?

Nicole Rauch unterstützt Kunden beim Herausarbeiten der Domäne einer Software mit Event Storming und bei der Umsetzung mit Domain-Driven Design. Sie legt auch selbst mit Hand an und entwickelt in TypeScript, Java oder in funktionalen Sprachen. Sie kennt sich in den Bereichen Compilerbau und formale Verifikationsmethoden aus und gibt im Interview Auskunft, wie man vom Bitschubser zum Algorithmiker wird.

Johannes Mainusch: Hallo Nicole, ich habe dich als Konferenzsprecherin, Trainerin und Senior-Programmiererin kennengelernt. Was machst du genau?

Nicole Rauch: Ich bin selbstständige Softwareentwicklerin und Softwareentwicklungs-Coach. Das bedeutet, ich entwickle selbst, mache Trainings und begleite Teams in der Softwareentwicklung. Meine Schwerpunkte dabei sind Domain-Driven Design und Event Storming, React und Redux, also Frontend-Programmierung, weiter zu funktionaler Programmierung und auch zu Backend-Entwicklung, Clean Code und Software Craftsmanship.

Was ist aktuell dein technisches Leibgericht?

Derzeit React und Redux mit TypeScript.

Okay, einmal kurz: Was sind TypeScript, React und Redux?

TypeScript ist eine Programmiersprache, die von Microsoft entwickelt wird. Sie basiert auf JavaScript, einer der verbreitetsten Programmiersprachen überhaupt. Nun hat JavaScript Nachteile, etwa den, dass die Sprache dynamisch typisiert ist. TypeScript hingegen ist statisch typisiert. Grundlegend gesagt legt ein Typsystem fest, welche Wertemenge einer Variablen zugeordnet werden darf, also zum Beispiel Strings oder Zahlen. Bei dynamisch typisierten Sprachen ist diese Information erst zur Ausführungszeit bekannt. Das heißt, wenn ein Fehler auftritt, zum Beispiel weil man versucht, zwei inkompatible Werte zu addieren, merkt man das erst in Produktion. Tests sind die einzige Möglichkeit, dies schon vorher herauszufinden. Und wenn man weiß, wie es bei JavaScript-Entwicklern um die Testmoral bestellt ist, erklärt das so einiges ...
Bei statisch typisierten Sprachen hingegen hat der Compiler die Typinformation bereits zur Entwicklungszeit und kann mir also schon beim Schreiben des Programms Hinweise geben. Oft kann ich das Programm gar nicht erst laufen lassen, wenn es solche Fehler enthält. Ein Teil der Tests kann also wegfallen, weil der Compiler einfach besser versteht, was ich in dem Programm tun will. Wenn ich in einem solchen Programm etwas ändere, zum Beispiel den Typ einer Variablen, dann kann mir der Compiler gezielt sagen, wo ich als Programmiererin Anpassungen durchführen muss, damit alles wieder stimmt.

„Entweder man diszipliniert sich beim Schreiben seiner Tests oder man benutzt diszipliniert ein Typsystem“

Unter den versierten Entwicklern geht der Streit ja noch viel tiefer. Diese sagen bei dynamisch typisierten Sprachen: „Das ist doch toll! In xy muss ich nicht so viel Boilerplate-Code schreiben wie in Java, und dann schreibe ich halt Tests für den Teil“. Und jetzt sagst du, dass Tests auch noch nicht so super sind.

Man braucht beim Programmieren nun mal Disziplin. Entweder man diszipliniert sich beim Schreiben seiner Tests, dafür ist man beim Programmieren freier. Oder man benutzt diszipliniert ein Typsystem, das empfinden viele als Einschränkung. TypeScript schlägt die Brücke, weil es einerseits erlaubt, weiterhin flexibel und leichtgewichtig genau wie in JavaScript mal eben ein Objekt zu schreiben, ohne dafür einen Typ angeben zu müssen. Und trotzdem habe ich andererseits an vielen Stellen auch statische Typen, die schon während der Programmierung eine automatische Überprüfung auf Fehler möglich machen.

Weil TypeScript über die sogenannte Typinferenz so schlau ist, viel über den Typ einer Variablen herauszufinden, ohne dass man die Typen immer explizit hinschreiben muss?

Teilweise muss man den Typ schon explizit hinschreiben. Beispielsweise bei

Schnittstellen, wenn man in andere Module hineingreift oder eine Funktion aufruft. Aber häufig kann man darauf verzichten. Da hilft mir dann tatsächlich die Typinferenz, weil sie aus einem Wert schon den Typ erkennen kann, zum Sting oder Objekt. Das ist im Gegensatz zu Java sehr elegant. In Java müsste ich dafür hinschreiben: Ich mache jetzt eine Klasse, mit dieser oder jener Ausprägung, und dann erst kann ich ein Objekt dieser Klasse erzeugen.

„In TypeScript hat man Leichtgewichtigkeit und trotzdem eine ziemlich gute Sicherheit“

Bleibt also die Freude der dynamisch typisierten Sprachen in TypeScript erhalten und man bekommt die gleiche Sicherheit wie in Java?

Im Großen und Ganzen stimmt das. Jedoch stelle ich hin und wieder fest, dass die Typinferenz, also die automatische Bestimmung des Typs einer Variablen, dann doch nicht so gut greift, wie ich mir das vorstellen würde. An manchen Stellen muss man den Typ einer Variablen doch noch mal hinterher schubsen. In TypeScript finde ich die Balance sehr angenehm: Leichtgewichtigkeit und trotzdem eine ziemlich gute Sicherheit. Nicht so gut wie in Java, aber ziemlich gut.

Stellt TypeScript dieses unglaublich verbreitete JavaScript auf bessere Schienen?

Ja, denn TypeScript wird komplett in JavaScript übersetzt und ganz normal ausgeführt. Man kann TypeScript daher überall dort einsetzen, wo auch JavaScript verwendet werden kann.

Vorhin hast du noch React und Redux erwähnt. Sind das Frameworks?

React ist eine reine Frontend-Bibliothek, das heißt, sie wird dazu benutzt, die Benutzeroberfläche zu gestalten. React hilft dabei, diese in Komponenten zu zerlegen und bei Bedarf mit Logik zu versehen. Beispielsweise, wenn ich bei der Online-Pizzabestellung die Anzahl der Pizzen im Warenkorb anzeigen möchte. Man würde allerdings kein React benutzen, wenn man eine normale Website mit statischem Text baut.

React fällt also in die Kategorie wie Vue.js von Evan You oder AngularJS von Google? Das sind die drei großen Frontend-Libraries/Frameworks, mit denen heute moderne Frontends erstellt werden.

Nicole Rauch

Funktion: selbstständige Softwareentwicklerin und Softwareentwicklungs-Coach

Mission: einfache, elegante Technik, die den Einsatzzweck der Software optimal unterstützt

Genau. React ist von Facebook und wird natürlich auch von Facebook selbst für ihr Produkt eingesetzt. Damit kann man sicher sein, dass React immer gut funktioniert.

Unglaublich verbreitet, aber bislang nicht unter einer Open-Source-Lizenz, oder?

Bis 2017 gab es eine Klausel im Lizenzvertrag, die besagte: „Wenn du ein Konkurrenzprodukt zu unserem baust, dann erlischt die Lizenz“. Diese Einschränkung in der Lizenzierung war ein riesiger Streitpunkt. Seit Version 16, die vor gut eineinhalb Jahren veröffentlicht wurde, hat React mit der MIT-Lizenz nun eine ganz normale Open-Source-Lizenz.

Was macht Redux?

Redux ist eine sehr kleine Bibliothek, die nur aus ungefähr 100 Zeilen Code besteht. Zweck von Redux ist es, im Frontend „State“ zu verwalten. Also Daten an einer Stelle zentral zu speichern. Denn in einer Applikation, welche Logik beinhaltet, benötige ich auch Daten.

Ist das dann, mit nur 100 Zeilen, eine Vereinheitlichung über Browser oder was macht Redux?

Ja, und es spielt unheimlich gut mit React zusammen, indem es React die Daten übergibt und aus React heraus kann man diese Daten sehr einfach aktualisieren, indem man Redux sagt, was man geändert haben möchte. Der Macher von Redux, Dan Abramov, ist überwältigt von der großen Resonanz, die er bekommen hat. Er hat es damals eigentlich nur als Demonstrator für eine Konferenz geschrieben, als kleines Gimmick. Aber viele Leute haben das gesehen und gedacht: „Cool, das kann ich gut gebrauchen“, und dann ist es viral gegangen.

Das heißt, es gibt tatsächlich noch diese „golden Nuggets” in der Programmierung, wo jemand mit nur 100 Zeilen Programmcode weltweiten Ruhm erlangen kann?

Ja! Das ist ein Beispiel dafür. Dan Abramov hat Redux erst 2015 geschrieben.

Nun habe ich ja am Anfang des Interviews funktionale Programmierung erwähnt. Wir sind fast alle mit objektorientierter Programmierung groß geworden. Wo ist der Unterschied?

Funktionale Programmierung bezieht sich auf Funktionen. Objektorientierte Programmierung bezieht sich auf Objekte.

Aber ich habe doch in einem Objekt auch Funktionen. Die heißen da Methoden, oder? Was ist denn der Unterschied zwischen einer Funktion und einer Methode?

Eine Methode ist eine Funktionalität, die einem Objekt zugeordnet ist. Eine Funktion ist etwas Eigenständiges. Sie bekommt Dinge übergeben und gibt etwas zurück.

Früher, als wir C, Fortran oder Cobol programmierten, hatten wir auch schon Funktionsblöcke. Kann man also sagen, dass wir schon immer funktional programmiert haben?

Technisch ist es richtig, dass man auch in C mit Daten und Funktionen gearbeitet hat. Allerdings hängen der funktionalen Programmierung noch verschiedene andere Paradigmen an. Erstens Immutability, also Unveränderbarkeit. Das bedeutet, einmal gespeicherte Daten werden nicht wieder verändert.

Die Daten in der Funktion oder außerhalb?

Bei rein funktionaler Programmierung gibt es in der Funktion erstmal keine Daten. Ich kann Daten in eine Funktion reinreichen oder zurückgeben. Aber die Funktion hat in sich selbst keine Daten. Das ist auch der Unterschied zur Methode in der

Objektorientierung. Die Methode kann auf die Daten des Objekts zugreifen und hat somit eigene Daten zur Verfügung.

Immutability habe ich mir so erklärt: Ich rufe bei der Versicherung an, sage, dass sich meine Telefonnummer geändert hat, und dann würde der Sachbearbeiter einfach aufschreiben „Mainusch hat angerufen. Der hat eine neue Telefonnummer" …

Mh, so könnte man das machen. Wichtig ist, dass ich auch in der Lage sein muss, überall dort auf die neue Telefonnummer zu verweisen, wo vorher die alte Nummer bekannt war. Erstmal schaffe ich nur ein neues Datenelement, in dem die neue Telefonnummer steht. Das alte Datenelement ist noch da. Aber wenn es nirgends mehr referenziert wird, wenn es also keiner mehr benutzt, dann verschwindet es.

Also Immutability heißt, keine Änderungen an vormals erzeugten Daten?

Ich ändere die Daten nicht, sondern dupliziere diese quasi. Wenn ich an einer anderen Stelle auf die alten Daten Bezug genommen habe, sieht die andere Stelle weiterhin die alten Daten. Das heißt, ich kann der Applikation nicht einfach neue Daten unterjubeln. Ich baue jedes Mal ein neues Element, in dem die neuen Daten stehen. Dadurch bin ich erst in der Lage, in meiner Applikation die gleichen Daten von verschiedenen Stellen aus zu benutzen, also sogenanntes Data-Sharing zu betreiben. In der Objektorientierung ändert man nun standardmäßig immer die Daten, und wenn diese Daten „geshared" wurden, kann das zu falschem Verhalten in anderen Applikationsteilen führen. Das ist eines der großen Probleme, die man in der Objektorientierung hat.

Das heißt, in der objektorientierten Welt, wo ich gerne mal Programme mit einer Größe von einer Million Lines of Code habe, gibt es viele Situationen, wo schwer zu verstehen ist, was genau passiert, wenn ständig Daten verändert werden. In der funktionalen Programmierung darf eine Funktion keine bestehenden Daten verändern, sondern muss neue Daten erzeugen, richtig?

Richtig! Das ist, in meinen Augen, ein Idealziel der funktionalen Programmierung. Es gibt nicht so viele Sprachen, die das so strikt fordern. Es gibt oftmals irgendwelche Abweichungen – man muss seine Sprache schon genau anschauen und selbst auch diszipliniert damit umgehen.

Okay, also funktionale Sprachen, wie zum Beispiel TypeScript, fordern nicht, dass man so arbeitet, sondern ermöglichen dies nur. Gibt es auch strikt funktionale Sprachen?

Haskell beispielsweise ist sehr strikt. Man kann sich aber auch selbst maßregeln, auch in anderen Programmiersprachen: In Java zum Beispiel kann man einfach eine Klasse machen, die nichts an ihren Daten ändert, sondern immer ein neues Objekt erzeugt.

Das heißt, eigentlich ist funktionale Programmierung viel mehr ein Paradigma, das in einigen Sprachen mal mehr, mal weniger implementiert ist. Aber es widerspricht dem Konzept, das wir bisher kennen, nämlich einfach Daten ausradieren oder überschreiben.

Genau. Man muss allerdings vorsichtig sein, was das dadurch erzeugte Datenvolumen angeht, weil Sprachen wie Java oder C-Sharp im Gegensatz zu funktionalen Sprachen nicht darauf vorbereitet sind, dass man immutable arbeitet.

Zusammengefasst: Immutability ist eine wesentliche Eigenschaft in der funktionalen Programmierung und bedeutet, dass eine Funktion auf eine Eingabe hin neue Daten zurückgibt und keine bestehenden Daten in ihrem Funktionsrumpf verändert.

Genau.

Ist das alles?

Nein, das ist nicht alles. Eine andere Sache, die zuerst in den funktionalen Sprachen vertreten war, nennt sich Higher-Order-Functions oder Funktionen höherer Ordnung. Das bedeutet, dass ich einer Funktion eine Funktion übergeben kann oder dass sie eine Funktion wieder zurückgibt.

Ist das wie in der Mathematik, dass ich eine Funktion auf eine andere Funktion anwenden kann und darauf wieder eine Funktion anwenden kann?

Das kannst du auch in Java, an der Stelle, wo du es hinschreibst, machen. Aber was du zum Beispiel in Haskell machen kannst, ist, dass du einer Funktion eine andere übergibst und dann entscheidet die erste Funktion, an welcher Stelle die übergebene Funktion aufgerufen wird. Diese übergebene Funktion kann dreimal oder hundertmal aufgerufen werden oder gar nicht.

Das Übergeben ganzer Funktionen an eine Funktion habe ich zuerst in Ruby gesehen, dann in JavaScript …

Genau, das ist aber viel weiter verbreitet. Smalltalk zum Beispiel macht das auch mit anonymen Funktionsblöcken, in C hat man Funktions-Pointer. In funktionalen Programmiersprachen unterscheidet sich eine Funktion nicht von anderen Datentypen. Darum nennt man diese dann auch „First Class Citizens", also gleichberechtigte Bürger. Somit kann ich prinzipiell mit Funktionen das Gleiche machen wie mit Strings, Zahlen oder anderen Datentypen.

So, nun haben wir als Paradigmen der funktionalen Programmierung erstens Immutability und zweitens Higher-Order-Functions kennengelernt. Gibt es noch etwas, was für funktionale Programmiersprachen kennzeichnend ist?

Ja. Das dritte Kennzeichen funktionaler Programmierung ist die Behandlung von Seiteneffekten. Seiteneffekte sind alles, was in einem Programm mit der Außenwelt zu tun hat, also etwa ein Datenbankzugriff, das Werfen einer Exception oder ein Eintrag in ein Logfile. Und natürlich braucht man auch das, denn wie sonst könnte man Benutzerschnittstellen und Ein- und Ausgaben behandeln. Nur mit solchen seiteneffektbehafteten Dingen kann man überhaupt ein Programm sinnvoll ausführen.
Allerdings trennt man in der funktionalen Programmierung immutable Funktionen strikt von den seiteneffektbehafteten. In Haskell haben diese Funktionen sogar einen anderen Typ, damit man gleich erkennen kann, wo die Seiteneffekte auftreten. Diese Trennung vereinfacht die Testbarkeit und das Verständnis von Funktionen ungemein. In Haskell verwendet man für diese Seiteneffekte Monaden.

Aha, da ist sie wieder, die Monade. Das ist eine schöne Erklärung, auf diesem Weg zur Monade zu kommen. Damit du alles andere schön „pure" halten kannst, haben wir für Ein- und Ausgabe eine Extra-Konstruktion gemacht, die heißt Monade …

Es gibt unterschiedliche Monaden für die verschiedenen Arten der Ein- und Ausgabe. Datei lesen, Datei schreiben, auf der Konsole etwas ausgeben. Und das sehe ich dann, wie gesagt, auch in der Typsignatur, meine Funktion gibt dann also beispielsweise eine Monade zurück.

„Eine Monade ist eine einheitliche Behandlung verschiedener Effekte auf die Außenwelt oder aus der Außenwelt heraus"

Es gibt in der funktionalen Programmierung also drei Dinge: Immutability, Higher-Order-Functions und Seiteneffekte wie Ein- und Ausgaben, die zum Beispiel mit Monaden behandelt werden könne. Ist das nun alles?

Ja, im Wesentlichen schon. Das sind die Grundbausteine der funktionalen Programmierung. Was nun passiert, ist dass die Leute in der Ordnung noch mehr Ordnung schaffen wollen und ähnliche Dinge möglichst gleich behandeln wollen. Das nennt man Abstraktion.

„Ich finde es irritierend, dass die funktionale Programmierung in Deutschland noch so wenig Fahrt aufgenommen hat"

Abstraktion kennen wir ja auch aus anderen Programmiersprachen …

Richtig, aber in der funktionalen Programmierung wird das noch viel weiter getrieben. Das ist auch ein Stück des Vorwurfs gegen die „Hardcore" funktionalen Programmierer, dass sie nämlich so abgedreht sind, dass sie nur noch in ihren abstrakten Konstrukten unterwegs sind und überhaupt nicht mehr verstehen, was normale Programmierer bewegt. Eine solche Abstraktion ist beispielsweise die Monade. Hier werden verschiedene Effekte der Außenwelt auf die gleiche Art behandelt, auch wenn jeder Effekt sich von den anderen unterscheidet.

Du hast mich kürzlich inspiriert, funktionale Programmierung mit TypeScript zu probieren. Das hat bei mir zu einem echten Klick im Kopf geführt, als ich auf einmal merkte, wie gut ich nach kurzer Zeit meine eigenen Programme lesen kann. Ist dieser Klick im Kopf ein typischer Effekt, wenn man in die funktionale Programmierung einsteigt?

Genau, das ist ein typischer Effekt. Das „Klick" ist ein echter Kick. Es kann unter Umständen etwas schwieriger sein, bis der eigene Quellcode kompilierbar ist. Aber dann ist es sehr gut lesbar.

Bin ich mit meiner Euphorie zur funktionalen Programmierung in eine Blase hinein gesaugt worden, oder erleben wir gerade den Anfang eines Paradigmenwechsels in der IT?

Das hängt davon ab, wo du hinschaust. Ich glaube nicht, dass es sich um eine Blase handelt. Natürlich lernst du als Entwickler weiter und möchtest interessantere Dinge machen und das Niveau des Standard-Programmierers hinter dir lassen. Ich finde es irritierend, dass die funktionale Programmierung in Deutschland im Vergleich zu anderen Teilen der Welt noch so wenig Fahrt aufgenommen hat.

Also leben wir in Hinblick auf funktionale Programmierung in Deutschland etwas hinterm Mond?

Ja. Wenn du zum Beispiel nach London schaust, so wird dort viel mehr auf funktionale Programmierung gesetzt. Besonders bei schwierigeren Applikationen, wo viel Gehirnschmalz in Algorithmen fließt. Funktionale Programmiersprachen sind oft viel formaler, ein bisschen so wie Mathematik. Wenn man ein Programm in so einer formalen funktionalen Programmiersprache ausdrückt, kann der Computer viel besser überprüfen, ob sich irgendwo ein Fehler eingeschlichen hat, und dies gleich dem Entwickler zurückmelden. Also der gleiche Effekt wie beim Übergang von dynamischen zu statischen Programmiersprachen, nur noch eine Nummer krasser.

„Programme einer formalen funktionalen Programmiersprache kann der Computer viel besser auf Fehler überprüfen"

Klick im Kopf ist ja ein Synonym für freudiges Lernen. Wird man durch funktionale Programmierung auch zu einem Higher-Order-Programmierer?

Ja, ich glaube schon. Ich glaube, funktionale Programmierung ist ein Konzept, das einem sehr weiterhilft. Wenn man ein Programm in Java schreibt, ist man sehr operational unterwegs. Stets denkt man darüber nach, was man wo speichert, und man ist Schritt für Schritt unterwegs. In der funktionalen Programmierung schreibt man eher hin, wie das Ergebnis aussehen soll. Wie man dahin kommt, überlässt man ein Stück weit dem Compiler. Das ist Programmierung in großen Sprüngen und nicht in kleinen Schritten.

Vom Bitschubser zum Algorithmiker?

Ja, genau. Das ist übrigens auch genau das, was ich beispielsweise an React schätze. Da schreibe ich einfach deklarativ hin, wie es aussehen soll, wenn das nächste Mal gerendert wird. Wer schon einmal Frontend-JavaScript-Applikationen geschrieben hat, der weiß, was das ansonsten oft für eine Plackerei ist.

Was mache ich denn nun als IT-Manager, wenn ich lese, dass wir noch ein neues Programmierparadigma in unserer Firma in Erwägung ziehen müssen?

Als IT-Manager habe ich hoffentlich verstanden, dass lebenslanges Lernen nicht einfach nur eine Phrase ist, um neue Leute anzulocken, sondern dass das ein Teil unserer Profession ist. Man muss die Leute in die Lage versetzen, sich weiterzubilden, wenn sie das möchten. Es gibt bestimmt einige im Unternehmen, die daran interessiert sind, und denen sollte man dann auch die Möglichkeit geben, sich weiterzuentwickeln.

Nicole, vielen Dank für das Gespräch.

Das Interview führte ...

Dr. Johannes Mainusch
(johannes.mainusch@kommitment.biz)
Berater für Unternehmen, die Bedarf im Bereich IT, Architektur und agiles Management haben. Dr. Mainusch ist seit 2012 Mitglied der OBJEKTspektrum-Redaktion.