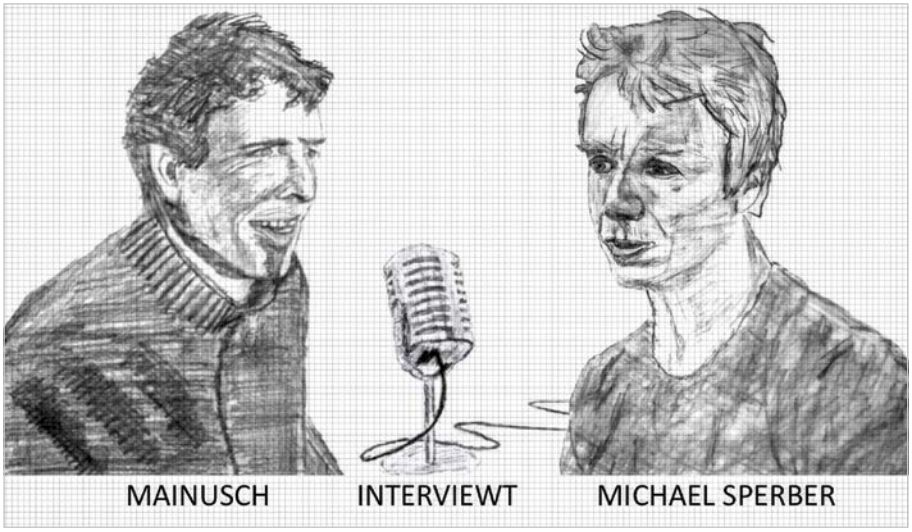
Interview mit Michael Sperber

Monaden – einfache Einheiten

OBJEKTspektrum sprach mit Dr. Michael Sperber über Monaden in der funktionalen Programmierung. Dr. Sperber unterstreicht: Monaden – wenn Du das ein paar Mal gemacht hast, dann schleicht sich das Verständnis von alleine an, ohne dass Du es merkst.



Johannes Mainusch: Immer, wenn ich mit Dir über funktionale Programmierung spreche, kitzelt es angenehm im Kopf. Aber das Gefühl, alles zu verstehen, bleibt oft aus. Also frage ich Dich einmal mehr, Michael, was ist eine Monade?

Michael Sperber: Monaden sind Dinge, die sich gut kombinieren lassen. Beispielsweise abhängige Berechnungen. Mein Lieblingsbeispiel ist: „Hoffnung" ist eine Monade. An ein kleines Kind habe ich eine einfache Hoffnung: Es soll lächeln. Eine andere Hoffnung ist, es soll seinen Brei essen. Beide kann ich kombinieren zu einer zusammengesetzten Hoffnung: Ich hoffe, dass das Kind erst den Brei ist und dann lächelt.

In der Reihenfolge?

Ja, manchmal hängt das zweite Ding halt vom ersten ab. Ich hoffe, das Baby lächelt, wenn ihm der Brei geschmeckt hat, also eine Hoffnung, die aus zwei Hoffnungen zusammengesetzt ist, die zweite abhängig von der ersten. Eine Monade ist eine abstrakte Eigenschaft konkreter Dinge. In Monaden kann ich Ketten aus kausalen Gliedern bauen.

Was aber ist nun daran so besonders?

Der Begriff wird halt häufig bei nichtfunktionalen Programmierern mystifiziert. Monaden gibt es überall, die Welt ist voll davon. Besser ist die Frage: „Was ist nützlich daran, diesen Begriff als Programmierer zu kennen?" Funktionale Sprachen machen den Umgang mit Monaden einfach. Die meisten anderen Sprachen sind nicht ausdrucksstark genug, um die Idee von Monaden zu repräsentieren.

„Eine Monade ist eine abstrakte Eigenschaft konkreter Dinge."

Kannst Du ein Beispiel geben, wo Monaden sehr nützlich sind?

Ich komme auf die Hoffnung zurück und auf eine Applikation aus der Produktionssteuerung. Dort kann alles Mögliche schief laufen. Etwa: Maschine geht kaputt oder ein Betriebsmittel ist verbraucht. Ich kann also immer nur hoffen, dass eine Maschine eine Anweisung ausführt, sicher kann ich nicht sein. Damit muss ich also auch die Hoffnung in meiner Software verwalten.

Also, das heißt, in Haskell/Javascript/Erlang sind solche Sachen einfacher als in einer rein sequenziell applikativen Programmiersprache wie etwa Java?

Ja. Also erstmal kurz zurück zur Hoffnung in der Produktionssteuerung: Ich hatte zunächst einmal alle speziellen Dinge direkt programmiert, die in der Produktion schief gehen können, und alle Kombinationen davon. Das wurde sehr kompliziert. Dann ging mir auf, dass Hoffnung eine Monade ist und dass meine programmierten Kombinationen nur Spezialfälle von Operationen in der Monadenbibliothek waren. So konnte ich mein altes Programm wegwerfen und habe Monaden verwendet.

„In Java gibt es auch Monaden, zum Beispiel in den Interfaces für Stream und Optional."

In funktionalen Sprachen ist das einfach, ich kann die Abfolge von Verarbeitungsschritten mit ungewissem Ausgang einfach als Verkettung von Monaden konstruieren. In Java gibt es ja auch Monaden, zum Beispiel in den Interfaces für Stream und Optional. Aber die Gemeinsamkeit zwischen diesen beiden Interfaces kann ich in Java nicht hinschreiben, das Typsystem ist nicht stark genug dafür.

Mike Sperber als Torvald Helmer in Nora im Theater U34

Wie bist Du zur funktionalen Programmierung gekommen? Und wie viel Aspirin hat das gebraucht?

1990 habe ich ein Buch über Computergrafik geschrieben. Und mein Co-Autor, Sebastian Egner, hat mir ein Buch empfohlen, „Structure and Interpretation of Computer Programs“ von Harold Abelson, Jerry Sussman und Julie Sussman. Das hat mein Leben verändert. Und Aspirin hat mir das jede Menge gespart.

Ein Compilerbuch?

Das Buch ist eigentlich berühmt (bekannt als das „Purple Book“). Es ist das Buch über die Anfängerausbildung am MIT. Ist jedoch als Anfängerbuch total unbrauchbar. Aber es basiert auf funktionaler Programmierung und zeigt sehr viele wunderbare Dinge, die man mit Abstraktion machen kann. Außerdem ist es toll geschrieben.

„Das Purple Book hat mein Leben verändert.“

Die gute Nachricht ist, funktionale Programmierung kann jeder lernen. Ich unterrichte seit 1987 Informatik. Damals als Austauschschüler in den USA. Später in der Uni habe ich darüber geforscht, wie man möglichst vielen Leuten möglichst gut programmieren beibringen kann.

Wie geht denn das?

Das Wichtigste ist, systematisch vorzugehen, anstatt nur Beispiele vorzustellen und dann hoffen, dass die Lernenden das durch Osmose auf ihre eigenen Probleme übertragen können. Dazu gehört insbesondere, dass jedes Konzept und jede Technik einen Namen bekommt, damit wir drüber reden können.

Die Herausforderung in Unternehmen ist doch häufig, dass wir Informatiker uns untereinander nur so lala verstehen, aber der Rest uns nicht versteht, oder?

Ich glaube, worauf Du hinaus willst, ist, „Monade” kann ich halt keinem Kunden sagen, richtig? Das habe ich auch mal gedacht. Ich habe aber festgestellt, dass es Kunden gefällt, mit solchen Ideen auf ihre Probleme loszugehen. Und das Wort finden die cool. Außerdem halte ich die funktionale Programmierung für die elegantere und bessere Methodik, um Probleme zu lösen. Erinnerst Du Dich noch an Deine Anfängervorlesung in Informatik – in den 80ern? Da gab es immer einen Abschnitt „Rekursion“, bei dem so Einige scheiterten. Man hat das dann in vielen Vorlesungen einfach aus dem Lehrplan geworfen. Das liegt aber nicht an der Rekursion an sich, sondern an der Didaktik dahinter.
Wenn Du die Frage stellst, „was ist Rekursion“, ist das schwierig trocken zu erklären. Sinnvoller wäre es, zu zeigen, welche Probleme man mit Rekursion lösen kann und wie. Und wenn Du das ein paar Mal gemacht hast, dann schleicht sich das Verständnis von alleine an, ohne dass Du es merkst. So ist das auch bei den Monaden: Du verstehst es schon, aber merkst es noch nicht.

Sprache in der IT ist eine der großen Hürden, die wir in der Produktentwicklung überwinden müssen. Kunden verstehen oft nicht, womit wir arbeiten. Ist das auch deine Erfahrung?

Erst einmal müssen wir den Kunden verstehen, das ist ja oft schon schwierig genug. Die kommen häufig aus einer fachlichen Domäne, die wir nicht notwendigerweise schon kennen. Eigene Wörter, domänenspezifische Begriffe. Das ist in der Regel die entscheidende Unterhaltung. Zunächst gilt es also, das verstandene Domänenwissen so zu wiederholen, dass der Kunde sich darin wiedererkennt.

... und am Ende des Tages wird daraus ein Programm. Also ist der Übersetzer von Domänenwissen in ein Programm immer der Informatiker?

Nein, im Idealfall ist eine Ebene in der Software eine domänenspezifische Sprache, die der Kunde selbst verwenden kann, um sein Domänenwissen zum Ausdruck zu bringen.

Also entsteht so Software richtigerweise von oben nach unten – von der Domäne hin zur Technik und nicht andersherum, von der Datenbank zum Kunden?

Ja, genau so. Wir finden beim Kunden häufig Software vor, die von unten nach oben gemacht ist und in der sich die Struktur der Datenbank in der Benutzerschnittstelle wiederfindet. Das sieht dann zum Beispiel so aus, dass in einer Wartungsoberfläche für Profi-Kaffeemaschinen das Benutzer-Interface für den Kunden ausschaut wie `phpmysqladmin`. Das ist dann halt nicht so gut.
Denn so wird der Kunde dazu gezwungen, sich den Restriktionen der Maschine zu unterwerfen, und bekommt seine Probleme oft nicht gelöst. Am schlimmsten habe ich das bei einem Enterprise-Dokumentenmanagementsystem gesehen, wo Suchanfragen für betriebswichtige Unterlagen mit regulären Ausdrücken in SQL übersetzt wurden. Um das System zu bedienen, mussten Benutzer das interne Datenbankschema kennen und das Übersetzungsschema für die Suchanfragen.

Dr. Michael Sperber

ist angewandter Theoretiker und Geschäftsführer der Active Group GmbH in Tübingen. Er ist international anerkannter Experte für funktionale Programmierung und wendet sie seit über 20 Jahren in Forschung, Lehre und industrieller Entwicklung an.

Motto: „Alles, was dir vor die Hände kommt, es zu tun mit deiner Kraft, das tu.” (Prediger 9:10)

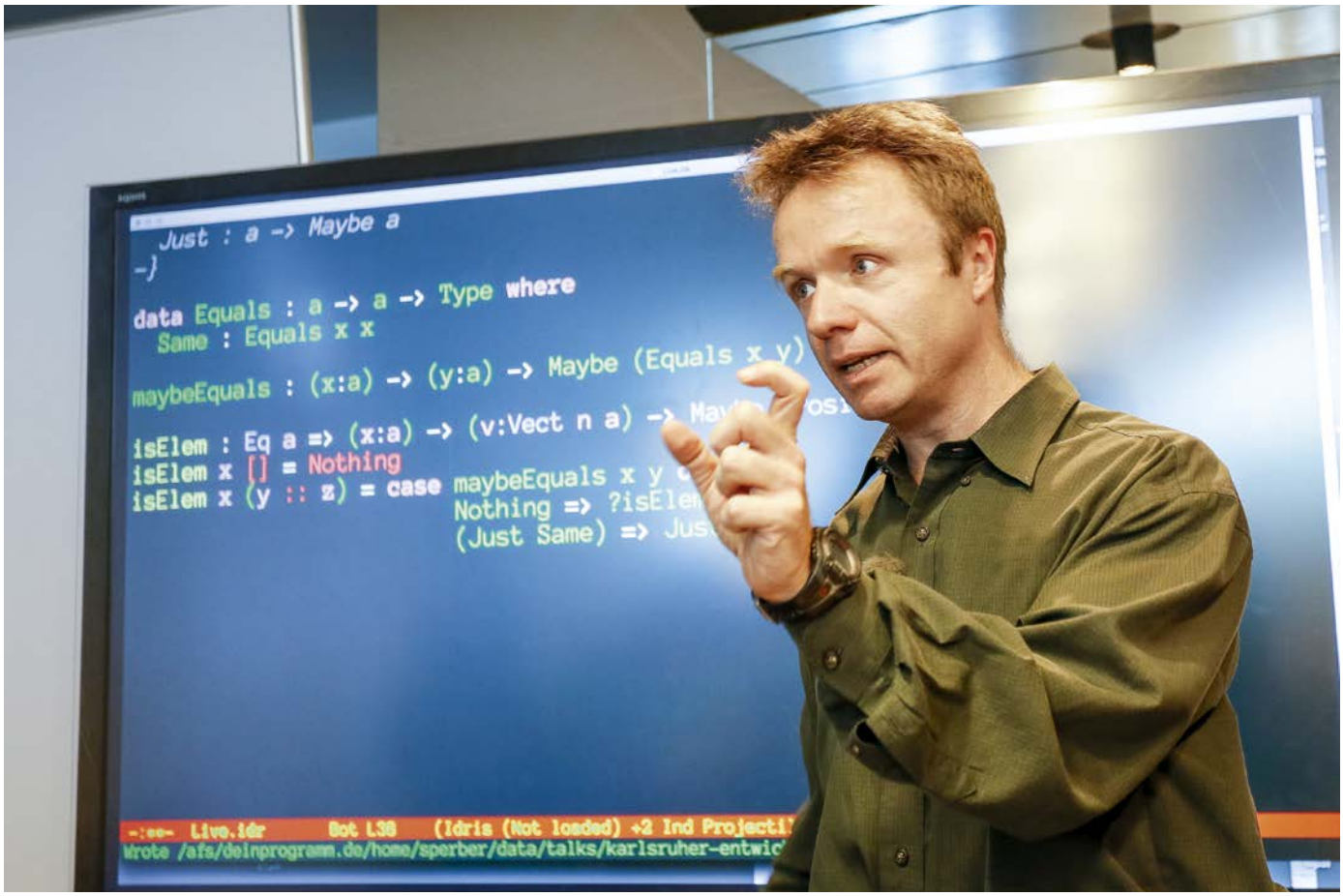
Lieblingswort: (liebevolle) Laubsägearbeit

Alter: 46 Jahre

Hobby: spielt leidenschaftlich freies Theater

Würde uns auch in der Kommunikation zwischen Kunden und Programmieren eine einfachere Sprache helfen? Und wie müsste die dann aussehen?

Einfachere Werkzeuge im Umgang mit der Sprache würden helfen. Etwa wenn Kunden die bei ihnen gültigen Business-Regeln, Vorschriften und Begriffe explizit benennen würden. Wir brauchen eben auch Sprache für Zusammenhänge und Regeln. Darum lohnt es sich, viel und ausführlich über moderne Programmierkonstrukte zu sprechen und hier auch unter Informatikern in den Diskurs zu gehen.



Entwicklertag Karlsruhe, 22. Mai 2017, Conference Day

Aber Programmiersprachen sprechen die Kunden doch eben gerade nicht? Wie soll das gehen?

Dabei helfen uns gerade diese domänenspezifischen Sprachen.
Viele Kunden benutzen ja auch Excel: Das ist keine besonders gute domänenspezifische Sprache, aber sie ist gut genug, dass viele sie gern verwenden. In vielen Domänen geht es aber deutlich besser. Eines der klassischen Beispiele aus der funktionalen Programmierung sind Finanzderivate, deren Beschreibung möglichst eindeutig sein muss. Die Idee einer domänenspezifischen Sprache dafür ist inzwischen erfolgreich von der Firma LexiFi kommerzialisiert und wird auch von Strukturierern in vielen Banken verwendet, also die Leute, die sich die Derivate ausdenken – nicht die Informatiker. Diese Programme sind direkt lauffähig, da muss also gar kein Informatiker mehr ran.

„Auch Excel ist eine domänenspezifische Sprache."

Gibt es Tools, um domänenspezifische Sprachen zu bauen, und ab welcher Domänengröße lohnt sich das?

In funktionalen Sprachen lassen sich domänenspezifische Sprachen oft leicht einbetten, das heißt, man schreibt eine Library für die Grundelemente der Domäne und benutzt die funktionale Sprache dann, um sie zusammenzubauen. Das geht sehr schnell, manchmal in wenigen Stunden. Es stimmt aber, in den ganz kleinen Projekten (weniger als zwei Monate) sind DSLs eher selten, ab da tauchen sie jedoch häufiger auf.

Wo werden sich DSLs in Zukunft durchsetzen?

Überall dort, wo Kunden sich in ihrem Business emanzipieren und nicht für jede Änderung einen Dienstleister beauftragen wollen.

Okay, ich bin inspiriert. Nun sag mir doch zum Abschluss noch einmal, wie ist nun die korrekte Definition einer Monade.

Eine Monade ist ein Tripel aus einem Typkonstruktor (so was wie „List", „Optional" oder „Stream") und zwei Operationen. Die erste Operation „unit" macht aus einem Wert eine monadische Berechnung, die diesen Wert produziert. Die zweite Operation „bind" kombiniert zwei monadische Berechnungen. Außerdem gelten drei mathematische Gesetze:

- Identität links: `(unit x) bind f = f x`
- Identität rechts: `m bind unit = m`
- Assoziativität: `(m bind f) bind g = m bind (λ x . (f x) bind g)`

Michael, 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.