# Moderne Zeiten

Mit ihrer Geburt unterliegen Technologien einem kontinuierlichen und sich immer weiter beschleunigenden Änderungsdruck. Voltaire hat sich einst einer italienischen Lebensweisheit bedient, als er davon sprach, das Bessere sei der Feind des Guten. Diese Erkenntnis passt auf Java und verwandte Technologien wie die sprichwörtliche Faust aufs Auge.

Neue Technologien adressieren anfangs einen überschaubaren Horizont, lösen dabei bestehende Probleme auf innovative Art, um irgendwann das Ende ihres Potenzials zu erreichen, was neue disruptive Technologiesprünge beziehungsweise Paradigmenwechsel auslöst. Dadurch kommen technologische Umwälzungen ins Spiel, die die Karten neu mischen. So zumindest die Theorie.

Und so verhielt sich das früher auch in der Praxis, als Produktzyklen noch in Jahren und nicht in Monaten gemessen wurden. Heutzutage ist das unrealistisch, da bereits zu viele Investitionen in existierenden Systemen & Infrastrukturen gelandet – und dort bisweilen auch versandet – sind. Anders ausgedrückt, innovative Fliehkräfte führen verstärkt zu einer Anpassung bestehender technologischer Lösungen. Technische Umbrüche erfolgen demzufolge mittlerweile von innen statt von außen. Das betrifft natürlich ganz speziell die Welt der IT.

Es gibt noch einen weiteren Punkt auf der Gesamtrechnung, den einige bisweilen übersehen oder unterschätzen. Als eine große Organisation vor fast anderthalb Jahrzehnten vor der Frage stand, entweder auf einen völlig neuen, innovativen Technologiestack oder auf eine Nachfolgetechnologie der bis dato verwendeten technischen Lösung zu setzen, fiel die Entscheidung zugunsten letzterer Option. Hauptmotivation für den damals gewählten Weg war die flachere Lernkurve der beschäftigten Entwickler und des Wartungspersonals. Gegenüber dem möglichen Innovationsschub durch technologischen Umbruch hatte der Faktor „Lernkurve" somit einen höheren Stellenwert. Die betreffende Organisation ist mit dieser Strategie übrigens sehr gut gefahren.

Lautet die Lösung also technologische Askese? Erinnern Sie sich noch an das Space-Shuttle-Programm der NASA mit seinen Technologien aus den Siebzigerjahren, die noch bis ins neue Jahrtausend überdauern mussten, da ein kompletter Umbruch weder ökonomisch noch politisch opportun gewesen wäre? Dort haben technische Schwächen und Workarounds zu „accidental complexity" geführt, was an dieser Stelle durchaus wörtlich gemeint ist. Nicht immer ist es daher zweckmäßig, an alten Technologien festzuhalten.

Wer jetzt als Informatiker milde darüber lächelt: Das lange im Umfeld von Apple-Betriebssystemen konkurrenzlose Objective-C gehört zu den Dinosauriern seiner Zunft. Immerhin entstand es in den frühen Achtzigerjahren, als Objektorientierung vor allem Mitarbeitern von Forschungslabors vorbehalten war. Ein weiteres Beispiel: Das heute führende Betriebssystem – gemessen an der Zahl seiner Distributionen und Installationen – heißt Unix. Ist es daher sinnvoll, das genannte Betriebssystem mit dem Space-Shuttle-Programm zu vergleichen? Mitnichten, denn Unix ist zwar im Grundsatz seinen architektonischen Grundsätzen treu geblieben, hat aber trotzdem eine gründliche Modernisierung erfahren, was Bedienoberflächen, Mikrokernelarchitekturen oder Formfaktoren betrifft. Deshalb würden wir heutzutage weder Linux noch Mac OS X zum alten Eisen zählen.

An dieser Stelle verkehrt sich übrigens die oft als Nachteil ins Feld geführte „Weichheit" von Software zu ihrer Stärke. Was für andere Disziplinen anachronistisch erscheint, kann für Softwareökosysteme durchaus funktionieren. Nehmen wir exemplarisch das Thema Scale-up versus Scale-out. Im Jahr 2007 veröffentlichte der damalige Marktführer Nokia zwei Dutzend (!!!) neuer Mobiltelefonmodelle für unterschiedliche Anwendungsprofile und Preissegmente. Im gleichen Jahr kam Apple mit dem iPhone auf dem Markt. Ein einziges Modell, aber dank App-Konzept eines mit enormer Flexibilität. Der Rest ist Geschichte.

Diesem Ansatz hat auch Java seinen nachhaltigen Erfolg zu verdanken. Das Ökosystem hat inzwischen über zwei Jahrzehnte auf dem Buckel. Zwischen Java 1 (1995) und Java 9 (2017) vergingen etwa 22 Jahre, zwischen Beethovens 1. und 9. Sinfonie rund 25 Jahre. Diesem Zeithorizonten lag jeweils ein kontinuierlicher Reifungsprozess zugrunde. Mit der Java-Plattform von anno 1995 lässt sich Javas Neunte deshalb nur noch ansatzweise vergleichen. Natürlich mag die Java-Plattform noch immer unter der ein oder anderen Kinderkrankheit leiden. Viele Probleme haben sich aber dank ständiger Verbesserungen verflüchtigt.

Java 9 offeriert wie bereits zuvor Java 7 und Java 8 umfangreiche Verbesserungen. Von Modularisierungskonzepten über eine REPL-Umgebung bis hin zu einer leichtgewichtigen JSON-Implementierung sowie einem HTTP/2-Client-API reicht das hinzugekommene Funktionsspektrum. All dies können wir unter dem Stichwort Produktivitätsschub verbuchen. Java 9 stellt meiner Meinung nach wesentlich mehr dar als bloße Kosmetik. Für Entwickler lohnt sich der frühzeitige Blick auf das neue Java allemal.

Grund genug, dem Thema in dieser und auch in nachfolgenden Ausgaben breiteren Raum zu geben. Ich hoffe, unser für diese Ausgabe kredenztes Menü trifft ganz Ihren Geschmack, wobei wir für Anregungen immer ein offenes Ohr haben.

In diesem Sinne viel Spaß bei der Lektüre wünscht

Ihr Prof. Dr. Michael Stal