Geleitwort

Wie inzwischen allgemein bekannt sein dürfte, habe ich JavaScript im Mai 1995 innerhalb von zehn Tagen entworfen – unter Stress und unter widersprüchlichen Anweisungen meiner Vorgesetzten: »Sorgen Sie dafür, dass es so aussieht wie Java!«, »Achten Sie darauf, dass es für Anfänger leicht erlernbar ist!«, »Stellen Sie sicher, dass es fast alles steuern kann, was es im Netscape-Browser gibt!«

Bei zwei wichtigen Aspekten habe ich mich von Anfang an um Richtigkeit bemüht (Funktionen erster Klasse und Objektprototypen), aber ansonsten bestand meine Reaktion auf die wechselnden Anforderungen und den unzumutbar engen Termin darin, JavaScript von Anfang an möglichst flexibel zu gestalten. Es war mir klar, dass Entwickler bei den ersten Versionen Patches anbringen mussten, um Fehler zu korrigieren, und bessere Vorgehensweisen auszuprobieren hatten als diejenigen, die ich in Form der mitgelieferten Bibliotheken zusammengeschustert hatte. Bei vielen Sprachen gibt es nur sehr eingeschränkte Möglichkeiten zur Veränderung. So ist es beispielsweise nicht möglich, integrierte Objekte zur Laufzeit zu bearbeiten oder zu erweitern oder Bindungen von Standardbibliotheksnamen durch Zuweisungen zu überschreiben. In JavaScript dagegen ist es möglich, fast jedes Objekt komplett zu ändern.

Ich glaube, dass ich angesichts der Umstände eine wohl ausgewogene Entscheidung für das Design getroffen habe. Gewiss führt dies in manchen Anwendungsbereichen zu Herausforderungen (beispielsweise bei der gefahrlosen Vermischung von vertrauenswürdigem und nicht vertrauenswürdigem Code innerhalb der Sicherheitsgrenzen eines Browsers). Allerdings war es unverzichtbar, ein sogenanntes Monkey-Patching zuzulassen, damit Entwickler Standardobjekte bearbeiten können, um sich um Fehler herumzulavieren oder um in älteren Browsern Emulationen von moderneren Möglichkeiten bereitzustellen (wie es bei der sogenannten Polyfill-Bibliothek der Fall ist, die man in Deutschland wohl eher »Moltofill-Bibliothek« genannt hätte).

Neben diesen teilweise profanen Verwendungszwecken hat die Formbarkeit von JavaScript auch verschiedene kreative Weiterentwicklungen durch die Benutzergemeinde ermöglicht. Führende Benutzer haben Toolkit- und Framework-Bibliotheken nach dem Muster anderer Sprachen erstellt – Prototype nach dem Vorbild von Ruby, MochiKit nach Python, Dojo nach Java und TIBET nach Smalltalk. Schließlich kam die Bibliothek jQuery (»New Wave JavaScript«), die für mich bei meiner ersten Begegnung im Jahr 2007 wie ein Nachzügler wirkte, und eroberte die JavaScript-Welt im Sturm, indem sie es vermied, sich an älteren JavaScript-Bibliotheken und damit an anderen Sprachen zu orientieren. Stattdessen bohrte sie das »Abfragen-und-umsetzen-Modell« des Browsers auf und vereinfachte es radikal.

Führende Benutzer und ihre Innovationsnetze haben damit ein »Eigenbau-JavaScript« geschaffen, das nach wie vor in anderen Bibliotheken nachgebildet und vereinfacht wird und auch den Bemühungen zur Standardisierung im Web unterworfen wird.

Im Rahmen dieser Entwicklung ist JavaScript abwärtskompatibel geblieben (wobei manche statt von »backward« von »bugward« sprechen, also »mit den alten Fehlern kompatibel«) und natürlich immer noch von Haus aus veränderbar. Das gilt auch noch nach der Ergänzung um bestimmte Methoden in der letzten Version des ECMAScript-Standards, die Objekte gegen Erweiterung und Objekteigenschaften gegen das Überschreiben schützen. Die Entwicklung von JavaScript ist aber noch lange nicht abgeschlossen. Wie bei jeder lebenden Sprache und jedem biologischen System ist Veränderung langfristig die einzige Konstante. Ich kann mir nicht vorstellen, dass es jemals eine einzige »Standardbibliothek« oder einen Programmierstil geben wird, der alle anderen ungültig macht.

Keine Sprache ist frei von Eigenheiten oder so streng, dass man universell gültige empfehlenswerte Vorgehensweisen dafür aufstellen könnte, und JavaScript ist eher das Gegenteil. Mehr als bei den meisten anderen Programmiersprachen müssen JavaScript-Entwickler daher einen guten Stil, eine korrekte Anwendung und empfehlenswerte Vorgehensweisen studieren und anwenden, um wirkungsvollen Code zu schreiben. Allerdings glaube ich, dass es ganz wichtig ist, nicht überzureagieren und strenge oder gar dogmatische Richtlinien aufzustellen.

Dieses Buch verfolgt einen ausgeglichenen Ansatz der auf belegbaren Problemen und konkreten Erfahrungen beruht, ohne strenge und übergenaue Vorschriften zu machen. Für viele Menschen, die versuchen, effektiven JavaScript-Code zu schreiben, ohne Ausdrucksstärke und die Offenheit für neue Ideen zu opfern, wird dieses Buch eine entscheidende Hilfestellung und ein treuer Begleiter sein. Es ist sehr zielgerichtet geschrieben, leicht zu lesen und bietet hervorragende Beispiele.

Ich habe das große Vorrecht, David Herman seit 2006 persönlich zu kennen, als ich im Auftrag von Mozilla zum ersten Mal Kontakt mit ihm aufnahm, um ihn als gern gesehenen Experten in das Standardisierungskomitee der Ecma aufzunehmen. Davids tiefe und allürenfreie Fachkompetenz und seine Begeisterung für JavaScript machen sich auf jeder Seite bemerkbar. Bravo!

Brendan Eich