NORMEN IN DER PROGRAMMIERUNG – ELEMENTE STATT SPRACHEN

DER AUTOR

Bernd Höhne
ist gelernter Industriemeister Technik mit anschließendem, abgeschlossenem Wirtschaftsingenieursstudium. Seit 1972 befasst er sich mit Computertechnik und ist in der Informationsverarbeitung, u.a. mit maschinennaher Programmierung von militärischen und industriellen Computeranlagen, Software-Engineering und -Entwicklung logistischer Systeme und Geschäftsanwendungen, tätig.

Haben Sie schon einmal gehört, dass Programmier/innen in C++ gescherzt hätten? Sicher nicht. Man unterhält sich nicht in C++, Java oder Basic. Warum? Nun, Programmiersprachen sind keine Sprachen. Zweideutigkeiten werden explizit vermieden und damit fehlt die Basis zum Scherzen. Es handelt sich bei Programmiersprachen vielmehr jeweils um ein Set von Anweisungen für Compiler oder Interpreter, die diese weiterverarbeiten und als Maschinenanweisungen für die Hardware, den Computer, aufbereiten. Bei Programmier-„Sprachen" handelt sich also um technische Definitionen - die dazu dienen, Computer zu steuern – und nicht um Sprache. Wissen tun wir, die IT-Schaffenden, das alle – aber ist es uns auch bewusst? Und wo liegen die Vor- und Nachteile dieser Betrachtung?

Der Umgang mit Programmiersprachen ist teilweise ideologisch angehaucht bis hin zur Mystifizierung. Das Anpreisen ihrer Fähigkeiten und die Diskussionen um ‚die richtige Sprache' liefert Anzeichen dafür, dass es von Zeit zu Zeit nötig tut, dieses Thema betont technisch und nüchtern zu betrachten. Die Normen zu Programmiersprachen bieten dazu Anlass. Sprachen werden ständig überarbeitet, demnächst steht eine umfassende Überarbeitung der vielgebrauchten Sprache C++ ins Haus. Bei solchen Überarbeitungen werden auch bestehende Elemente immer wieder neu definiert, was Kompatibilitätsprobleme und Nacharbeitung mit sich bringt. Wieso eigentlich? DIN A4 wird ja auch nicht neu festgelegt, nur weil ein neues Papierformat oder eine neue Fertigungsmethode dazu kommt.

Ingenieure und Techniker arbeiten mit einer Vielzahl genormter Basiselemente zur Konstruktion von Geräten und Anlagen wie zum Beispiel bei Transistoren, Kondensatoren, Spulen, Schwingkreisen, Widerständen und vielem mehr. Keiner käme auf die Idee, Widerstände neu zu ordnen, nur weil Drehwiderstände dazu kommen. Trotz der Erfindung von Thyristoren blieb die Definition von Schaltbild und Funktion des Transistors und anderer Bauelemente davon unberührt. Technische Definitionen zu einer Sprache zusammenzufassen, hat man in der Elektrotechnik schlicht verpasst – oder war und ist das vielleicht gar nicht nötig? Eine vielleicht gar unnötige Verkomplizierung? Dabei liegt der abstrakte Vergleich zwischen definierten Konstruktionselementen (z. B. von Elektroingenieuren) und Objekten einer Programmiersprache auf der Hand.

Mathematiker haben ihre Disziplin, die ja Ingenieuren und Informatikern nicht ganz unbekannt ist, seit Hunderten von Jahren weiterentwickelt und viele neue Techniken, Methoden und Paradigmen sind dazu gekommen. Aber auch die Mathematiker definierten nicht ihre Grammatik und Symbolik neu – nur weil neue Methoden und Verfahren dazu kamen. Ein Gleichheitszeichen ist ein Gleichheitszeichen bleibt ein Gleichheitszeichen.

Anders in der Informatik. Neue Programmiermethoden und Techniken werden bevorzugt in neuen Sprachen ihrer Bestimmung zugeführt. Dabei wird auch erst einmal mit allem aufgeräumt, was dem Erschaffer der neuen Sprache an alten Sprachen nicht gefallen hat. Prototypen, Definitionen, Deklarationen, Kommentarzeichen, Zuweisungen, Bitoperationen – man kann so vieles verschönern. Dabei kann ich mich des Eindrucks nicht erwehren, dass Eitelkeiten oft eine größere Rolle spielten als technische Notwendigkeiten.

Die Nachteile neuer Programmiersprachen liegen dabei auf der Hand. Die Neufestlegung der Elemente, der Objekte, der Schlüsselworte, ihres Zusammenspiels und der Grammatik ist in der Leistungskette der Informationstechnik eher unproduktiv. Der Fortschritt wird gebremst, weil sinnvolles Neues mit Nacharbeiten und Umbauten an bestehenden Systemen erkauft werden muss, will man Bewährtes mit Neuem verbinden. Aber nicht nur Kompatibilität, und Zuverlässigkeit leiden. Die Systeme werden zudem träge, weil immer mehr Softwareschichten und Bibliotheken zur Verknüpfung unterschiedlicher Sprachsysteme und der damit implementierten Lösungen das System aufblähen.
Weiter müssen Programmierer und Systemanalytiker immer wieder auch Dinge neu lernen, die sie eigentlich schon kennen. Warum musste man in den 80ern komplett neue Syntax lernen nur um die Vorteile von Objekten nutzen zu können?

Die aufgezeigten Nachteile bieten Anlass darüber nachzudenken, was man ändern könnte, um mit Fortschritten in der Softwaretechnik so produktiv und effizient umzugehen, wie Ingenieure das mit Hardware tun – vom Toaster bis zum Raketenmotor.

Man könnte die Syntax der technischen Anweisungen und Objekte einzeln normen. Nicht die Sprache sondern die Elemente der Syntax würden genormt. Ein Schlüsselwort ‚int2' könnte für einen Integer der Länge der Größe von 2 Byte stehen – in jedem System. ‚/*' könnte als Kommentareinleitung genormt sein wie ‚*/' als Kommentausleitung. Ein Programmblock könnte mit dem ‚{, und ‚}' als Beginn- und Ende-Zeichen fest genormt sein – für alle Sprachen. Die Normung der Elemente machte die Normung der Sprachen überflüssig. Die Programmiersprachen in ihrer heutigen Form wären überflüssig und könnten einer weltweiten genormten Syntax-Bibliothek weichen.

Wenn Objekte z. B. dazu kommen, legt man fest, wie die Syntax dazu aussieht.

Einfache Objekte, komplexe Objekte, mit und ohne Vererbung haben dann selbstverständlich andere verbindliche Syntax-Festlegungen.

Vorteile, die aus der Ingenieurtechnik kommen. Denn selbst in einem japanischen Schaltplan kann ein französischer Elektroingenieur mühelos die Bauteile erkennen und damit ihre Funktion. Ein Transistor ist ein Transistor, eine Diode ist eine Diode – egal ob in der elektrischen Zahnbürste oder in einem Kraftwerk. Und eine neue Leuchtdiode, die mit Hochleistung weißes Licht ausstrahlt, kommt einfach mit einer neuen Darstellung dazu, die für ihre besondere Beschaffenheit steht.

Kein „Umlernen" mehr - nur noch „Dazulernen". So funktionieren Operatoren immer gleich. Ein verlockender Gewinn, wie mir scheint. Gerade weil vieles in unterschiedlichen Programmiersprachen ähnlich ist, aber im Detail anders funktioniert – ist Software so unzuverlässig und fehleranfällig.

Die Anzahl der Elemente – Schlüsselbegriffe - würde zwar deutlich steigen, aber deren Funktion wäre eindeutig. Elemente, die aus der Mode kommen werden eben nicht mehr benutzt, sie bleiben im Regal. Sofern alte und neue Elemente in einer Anwendung zusammenspielen, bleibt die Anwendung sicher, denn die Funktionalität wäre unmissverständlich bekannt und erprobt.

Der Compilerbau würde sicher anspruchsvoller, böte Differenzierungspotenzial ähnlich einem CAD-System. Der Compilerbauer muss dann entscheiden, welche Elemente, Schlüsselworte, Objekte (also Bauteile) er berücksichtigt und welche er als Erweiterung zum Produkt anbietet. In jedem Compiler besitzt dann aber ein einfaches Objekt (z. B. ohne Vererbung), die gleiche Syntax und die dazugehörige Funktionalität – unverwechselbar und verlässlich. Dass ein Integer ‚INT' in alten Software-Systemen 2 Byte groß ist, in anderen Systemen dann 4 Byte - was erhebliche Auswirkungen auf die interne Vektor-Arithmetik hat - fällt als Fehlerquelle aus, denn das eine Sprachobjekt ‚INT2' hätte eine andere Syntax als ‚INT4', um ein einfaches Beispiel zu geben. Fehler solch banaler Art führten beispielsweise zum Absturz der Rakete Ariane V und wären mit klaren Normen auf der Elementenebene statt der Normung ganzer Sprachen wesentlich leichter zu vermeiden.

Natürlich muss man sich auch fragen, warum – wenn der Gedanke nicht grundsätzlich falsch ist - gibt es das nicht längst. Wo doch die Vorteile auf der Hand zu liegen scheinen und Hardwaresysteme immer noch deutlich innovativer und vor allem zuverlässiger sind als Software.

Möglicherweise liegt es am Marketing. Anders als bei Hardware, vom Toaster bis zum Auto, wird die unbestrittene Flexibilität von Software auch zum Schutz und zur Eroberung von Märkten eingesetzt. Schnittstellen, die nicht passen zwingen zum Systemwechsel oder dem Beibehalten des Altsystems. Es gibt bis heute keine verbindliche, stabile SQL-Schnittstelle zur Kommunikation zwischen Applikation und Datenbank. Kleine Unterschiede sollen den Herstellern den Markt erhalten oder erobern. Schlimmer noch die Programmierelemente, die Internetbrowser steuern. Was im IE geht, geht im FIREFOX noch lange nicht zuverlässig und umgekehrt. Zur Lösung wird eine Menge Aufwand getrieben, die Systeme uni-funktional zu gestalten. Eine andere Variante ist, den Browser so einzustellen, dass er sich verhält als wäre er ein anderer Browser. Dann tut ein OPERA so, als wäre er ein FIREFOX. Gegen solche Emulationen wehrt sich gerade Apple, weil es die Systeme unzuverlässig macht. Warum so zaghaft Mr. Jobs? Warum nicht gleich zu richtigen Normen wechseln, anstatt bestimmte Programmiersprachen vorzuschreiben?

Fortschrittlich scheint das alles nicht. Normen dagegen klingen so altbacken. Die Normung der Kleinteile wäre aber ein echter Fortschritt in der Programmierung und könnte dazu führen, dass wir auf den ganzen Mittelware-Müll verzichten können, was uns zusätzlich den Vorteil bringt, der für uns alle so wichtig ist – Zuverlässigkeit. ■