8.9.92

Überflüssiger Streit um Programmiersprachen

Wirtschaftliche Überlegungen im Vordergrund

*Von Hartwig Thomas**

Immer wieder werden unter Programmierern heftige Kontroversen ausgetragen, welche die bessere *Programmiersprache* sei. Auftraggeber, Manager und sogar Anwender erzwingen die Verwendung der einen oder anderen Entwicklungsumgebung häufig auf Grund von falsch verstandenen «strategischen» Überlegungen. Überraschenderweise ist fast immer allgemein von Programmiersprachen und selten von speziellen *Implementationen* die Rede. Im folgenden soll die Frage der Programmiersprache bzw. der Entwicklungsumgebung aus der Position des *DOS/Windows-Entwicklers* zuhanden des Managements beleuchtet werden. Es geht hier um die rationalen, *strategischen Gründe* für den Management-Entscheid, die Programmierung im Betrieb auf eine gewisse *Umgebung* festzulegen. Im besonderen interessieren die Kriterien für die Wahl einer solchen Umgebung: Adäquatheit, Portabilität und Produktivität.

Die Programmiersprache Pascal . . .

Nachdem in den fünfziger Jahren Hochsprachen wie *Fortran* ad hoc erfunden worden waren, um den Programmierern die Arbeit zu erleichtern, bemühte sich in den Sechzigern eine Gruppe von Wissenschaftern um eine formale Spezifikation der endgültigen Hochsprache. 1968 entwickelte *Niklaus Wirth* an der ETH Zürich die Algol-inspirierte Programmiersprache *Pascal*, von welcher böse Zungen behaupten, das Hauptziel ihres Schöpfers habe darin bestanden, den Compiler möglichst einfach zu halten. Die Programmierer empfingen Pascal mit einem *Gelächter der Verachtung:* Die *strenge Typisierung* verunmögliche es, systemnahe Operationen auszuführen, ohne externe Fortran- oder Assemblerprogramme zu linken. Das Pascal-Konzept beruhe ausserdem auf der Vorstellung eines einzigen monolithischen Programms, was die modulare Entwicklung grosser Systeme stark behindere. Man hört diese Einwände heute noch. Dabei wird übersehen, dass sich Programmiersprachen weiterentwickeln. Schon um 1980 waren die gravierendsten Probleme aus dem Weg geschafft: Die mangelnde Modularität war von Wirth zum Anlass genommen worden, den Pascal-Nachfolger *Modula* zu entwickeln, dessen wichtigstes Konzept in verschiedenste Pascal-Dialekte (etwa *Microsoft Pascal* oder *Turbo Pascal*) integriert wurde.

Auf dem *PC* trat Pascal einen *Siegeszug* an. Die formale Definition und die Primitivität der Sprache erwiesen sich auf dem Mikrocomputer als wesentliche Stärken: man kann Pascal-Compiler entwickeln, die mit einem Minimum von Ressourcen (Disk, Speicher, Zeit) auskommen. Heute dürfte Pascal diejenige Sprache sein, für welche die *schnellsten Compiler* existieren. Dank diesem Grosserfolg musste sich Pascal denn auch die neueste Mode der «objektorientierten» Programmierung einverleiben.

. . . und die Programmiersprache C

Die *Programmiersprache C* wurde Anfang der siebziger Jahre von *Dennis Ritchie* für das Betriebssystem *Unix* als Nachfolger der Programmiersprache B entworfen und implementiert. Im Gegensatz zu Pascal verzichtete man auf eine formale Definition. Die Bedeutung einiger Konstrukte ist kontextabhängig und wegen der mangelnden formalen Definition von Compiler zu Compiler verschieden. Wegen dieser *Kontextabhängigkeit* ist jeder C-Compiler gezwungen, in mehreren Durchgängen zu kompilieren. Die hohe *Geschwindigkeit* eines kompakten Pascal-Compilers kann deshalb nur schwer erreicht werden. Auch C entwickelte sich im Laufe der Jahre. Während sich bei Pascal die verschiedensten Standardisierungsgremien (DIN, ISO . . .) damit beschäftigten, hinter der jeweiligen Realität hinterherzuhinken, führte beim C die *Standardisierung* durch das amerikanische Standardinstitut *ANSI* viele moderne Konzepte ein, welche der Norm einen hohen Beachtungsgrad verliehen.

Parallel mit Pascal absorbierte auch C die neue *Mode der «objektorientierten» Programmierung.* Unter dem Namen C++ soll der C-Compiler mit objektorientierten Erweiterungen ein (oder zwei) Plus haben gegenüber dem alten C. Ebenso wie bei den Objekt-Pascal-Dialekten handelt es sich bei C++ um eine *echte Obermenge* von C. Das heisst, alle alten C-Programme können auch unter C++ weiterverwendet werden, aber man hat neue Möglichkeiten der Gestaltung zur Verfügung.

Adäquatheit weitgehend gegeben

Eine grundlegende Frage beim Unternehmensentscheid für eine Entwicklungsumgebung ist die Frage nach der Adäquatheit. *Löst die Sprache* (alle) *unsere Probleme?* Grundsätzlich kann man von «höheren» Programmiersprachen wie Pascal, C, Fortran, PL/1 usw. sagen, dass sie weitgehend *gleich ausdrucksstark* sind. Man kann in ihnen (fast) alles programmieren, was in der betreffenden Umgebung (Betriebssystem und Hardware) möglich ist.

In gewissen Fällen ist auf der Zielhardware oder für das Betriebssystem kein guter *Compiler* der einen oder anderen Sprache verfügbar, oder die *Betriebssystemunterstützung* der Sprache ist mangelhaft. Es gibt Halbfabrikate zur Ansteuerung spezifischer Peripheriegeräte, welche die Verwendung eines spezifischen Compilers erzwingen. Unter solchen Umständen muss man bei der Wahl der Entwicklungsumgebung Kompromisse machen. Auf dem PC haben wir diese Probleme selten, da die wichtigsten Programmiersprachen vorhanden sind und ihre Hersteller jeweils auf Neuerungen recht schnell reagieren.

Portabilität oft überschätzt

Das häufigste Argument zur Rechtfertigung einer Entwicklungsumgebung ist das *Schlagwort der Portabilität.* Da man aus der Softwarekrise in der Chefetage gelernt hat, dass die *Programmentwicklung teuer* ist, fürchtet man den Substanzverlust, der eintreten würde, wenn man Uralt-Applikationen nicht mehr in eine neue Hardwareumgebung «portieren» kann. Professionelle Anbieter von Software leben davon, für immer *neue Systeme immer Neues* entwickeln zu können. Mit einer portierten Uralt-Anwendung käme man bei Kunden schlecht an. Der Markt zwingt dazu, jedes neue Betriebssystem, jede neue Konfiguration von Peripheriegeräten zu unterstützen. Werden Anwendungen heute auf Windows «portiert», verändern sich diese in einem so hohen Mass, dass höchstens ein Bruchteil der alten Codes verwendet werden kann.

Evident ist auch, dass Programmiersprachen bei weitem nicht so standardisiert sind, dass von Portabilität die Rede sein kann. Natürlich kann man sich dazu zwingen, permanent für mehrere

* Hartwig Thomas ist Geschäftsführer der Enter AG, Zürich.

Betriebssysteme *gleichzeitig* zu entwickeln. Der Programmierer ist dann gezwungen, sich auf das zu beschränken, was beiden Betriebssystemen gemeinsam ist. Wer etwa für DOS/Windows und MacIntosh entwickelt, darf nur beschränkt Multitasking-Eigenschaften einsetzen und in jedem Fall die schlechtere beider Welten wählen. Das portable Programm kann leicht dreimal so viel Aufwand kosten wie die separate Entwicklung in der angemessenen Sprache auf jeder der beiden Maschinen.

Für den Programmierer sind somit heute die *Wahl der Programmiersprache und ihre Standardisierung sekundär* geworden. Die wesentlichen Faktoren der Produktivität liegen beim Einkauf von *Halbfabrikaten,* im Aufwand für *Weiterbildung* und in der *Bequemlichkeit* der Entwicklungsumgebung. Im Gegensatz zur grossen Furcht vor dem Substanzverlust bei der Neuentwicklung alter Anwendungen zeigt sich in der Praxis, dass bei der Neuimplementation eines Problems, das man schon einmal gelöst hat, nur ein Bruchteil der Zeit verwendet wird, die es beim erstenmal gekostet hat. Die gesamte *Erfahrung mit der Lösung des Problems ist also «portabel».* Diese Portabilität ist allerdings nur transferierbar, wo die vergangene Erfahrung dokumentiert wird. Wem also Portabilität wirklich ein Anliegen ist, der sollte mehr Gewicht auf menschenlesbare Dokumentation der Programmierer-Erfahrung legen. Die Praxis und die Beobachtung der Marktentwicklung der letzten zehn Jahre hat uns gelehrt, dass es äusserst gefährlich ist, «strategisch» zu denken. Niemand kann sagen, welche Fähigkeiten, welche Werkzeuge und welche Produkte Programmierer in fünf Jahren einsetzen müssen. Vor drei Jahren hat kaum jemand die heutige Windows-Welle prognostiziert.

Gegen Vorurteile und Scheuklappen

Natürlich ist es notwendig, sich im kleineren Projektteam auf eine Entwicklungsumgebung zu einigen. In grösseren Firmen könnte man aber die geeignete Wahl der Entwicklungsumgebung ruhig an den *Ausbildungsstand und die Vorlieben der Programmierer anpassen,* wie man auch jedem anderen Handwerker die Freiheit zugesteht, seine Arbeitswerkzeuge selber zu bestimmen. Bei der Entscheidung für eine Entwicklungsumgebung prüfe man deshalb die Implementationen, die für das gewählte Betriebssystem, die Hardware und die Peripheriegeräte erhältlich sind, und ihre Zusatzwerkzeuge.