Unternehmensentscheid: Software-Entwicklungsumgebung

C oder Pascal?

Immer wieder werden unter Programmierern heftige Kämpfe ausgetragen, welches die bessere Programmiersprache sei. Auftraggeber, Manager und sogar Anwender geben oft nicht nur ihren Senf dazu, sondern 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 diese Frage der Programmiersprache bzw. der Entwicklungsumgebung aus der Position des DOS/Windows-Entwicklers zuhanden des Managements beleuchtet werden.

Es geht mir um die rationalen, strategischen Gründe für den Management-Entscheid, die Programmierung im Betrieb auf eine gewisse Umgebung festzulegen. Im besonderen interessieren die Kritierien für die Wahl einer solchen Umgebung: Adäquatheit, Portabilität und Produktivität.

Der Stand der Programmiersprachenvorurteile vor zehn Jahren wirft ein Schlaglicht auf die heutigen:
- *Pascal* eignet sich nur für einfache technische Anwendungen.
- Komplexe Programme muss man in *PL/1* schreiben.
- Für physikalische Berechnungen ist nur *Fortran* geeignet (Unter keiner anderen Programmiersprache gebe es brauchbare Numerikpakete und ausserdem könne man nicht ohne EQUIVALENCE und GOTO programmieren).
- Kommerzielle Anwendungen erfordern den Einsatz von *COBOL* (*Pascal* und *C* rechnen intern mit binären Zahlen und sollen darum nicht korrekt auf 5 Rappen runden können).
- Auswertung von technischen Messungen ist nur in *APL* möglich.
- *Basic* ist die natürliche Sprache der Mikrocomputer.
- *C* ist ein Hobby der Betriebssystemforschung und eignet sich nur für akademische Zwecke.
- Für KI-Anwendungen kommt nur *LISP* in Frage.

Damit auch Nichtprogrammierern ungefähr klar ist, wovon die Rede ist, gebe ich als erstes einen kurzen Überblick über die aus meiner Warte heute wichtigeren Programmiersprachen für DOS/Windows.

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 Wissenschaftlern um eine formale Spezifikation der endgültigen Hochsprache.

Um 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 systemnahe Operationen auszuführen, ohne externe Fortran- oder Assemblerprogramme zu linken. Das *Pascal*-Konzept beruhe ausserden auf der Vorstellung eines einzigen monolithischen Programms, was modulare Entwicklung grosser Systeme stark behindere.

Man hört diese Einwände heute noch. Dabei wird systematisch übersehen, dass sich Programmiersprachen immer weiter entwickeln.Schon um 1980 hatten sich Programmierer und *Pascal* soweit entwickelt, dass die gravierendsten Probleme aus dem Weg geschafft waren: Nach zehn Jahren Üben schien es uns nicht mehr unnatürlich, ohne GOTO zu programmieren. Die Dinosaurier-File-systeme waren meist durch lineare Datenströme ersetzt worden, wo man mit *read*, *write* und *seek* ganz gut zurecht kam. Die mangelnde Modularität war von Wirth zu 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* eine späten 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 „objekt-orientierten" Programmierung einverleiben.

Nicht ganz zufällig fällt die Einführung von „Objekten" zeitlich zusammen mit dem Durchbruch der „graphischen Benützerschnittstellen". Zusammen mit Windows wurde uns daher mit dem Paket *Turbo Pascal für Windows* (TPW) eine reichhaltige Bibliothek von Windows-Objekten beschert, welche die Entwicklung von Windows-Anwendungen wesentlich vereinfacht.

C

Die Programmiersprache *C* wurde anfangs 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. *ANSI C* ist von fast allen Unklarheiten befreit und hat sich in vielen Punkten an *Pascal* angenähert.

Während *Pascal* an den europäischen Universitäten als erste Computersprache der Informatikstudenten dominierte, nahm in den Vereinigten Staaten *C* diese Rolle ein.

Etwas später als *Pascal* erschienen die ersten brauchbaren *C*-Compiler auf dem PC. Da die Entwickler der neueren Betriebssysteme von Mikrocomputern diese alle selber in *C* programmieren (AmigaDOS, Windows, OS/2, . . .), kommen neuerdings häufig die ersten Schnittstellen zum System in dieser Sprache auf den Markt.

Parallel mit *Pascal* absorbierte auch *C* die neue Mode der „objekt-orientierten" Programmierung. Unter dem Namen *C++* soll der C-Compiler mit objekt-orientierten 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 noch unter *C++* weiterverwendet werden, aber man hat neue Möglichkeiten der Gestaltung zur Verfügung. Ein oft gehörter Stossseufzer, man müsse nun alles Erarbeitete wegwerfen und neu anfangen, ist also in beiden Fällen nicht berechtigt.

Auf dem PC haben wir eine sehr brauchbare Objektbibliothek (OWL = Object Windows Library) von Borland für die Entwicklung von Windows-Anwendungen zur Verfügung. *C++* von Borland und *Turbo Pascal 6.0* sind von ihren ursprünglichen Vorbildern *Standard-Pascal* und *Kernighan & Ritchie-C* weiter entfernt als voneinander. Die Bevorzugung der einen oder anderen Umgebung wird immer mehr eine Frage des Geschmacks und der Vertrautheit.

Adäquatheit

Eine grundlegende Frage für den 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 aber 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

Das häufigste Argument zur Rechtfertigung einer Entwicklungsumgebung ist das Schlagwort der Portabilität. Da man aus der Softwarekrise in der Cheftetage gelernt hat, dass Programmentwicklung teuer ist, fürchtet man den Substanzverlust, der eintreten würde, wie den Beelzebub, wenn man Uralt-Applikationen nicht mehr in eine neue Hardwareumgebung „portieren" kann.

Mit dem Einzug des PC und dem Absterben der Mainframes und ihrer EDV-Abteilungen hat sich eine neue Arbeitsteilung herausgeschält: die Konzerne sehen ein, dass sie mit selbstgestrickter Software nicht mit professionellen Software-Anbietern konkurrenzieren können. Auch wenn jede Grossbank eine Kantine besitzt muss sie deswegen nicht noch eine Grossbäckerei führen. Sie kann ihre Brötchen beim Bäcker einkaufen.

Als professioneller Anbieter von Software leben wir davon, dass wir für immer neue Systeme immer Neues entwickeln können. Mit einer portierten Uralt-Anwendung kämen wir bei unsern Kunden schlecht an. Der Markt zwingt uns dazu, jedes neue Betriebssystem, jede neue Konfiguration von Peripheriegeräten zu unterstützen. Wenn wir also unsere Anwendungen heute auf Windows „portieren", verändern sich die Anwendungen in einem so hohen Ausmass, dass höchstens ein Bruchteil des alten Codes verwendet werden kann.

Wir haben auch gelernt, dass Programmiersprachen bei weitem nicht so standardisiert sind, dass von Portabilität auch nur im Entferntesten die Rede sein kann. Da die Standardisierung bei der Behandlung der Eigenheiten des Betriebsystems durch die Programmiersprache aufhört, muss man bei der Portierung „einfach" alle nicht-standard Funktionen der einen Implementation auf dem anderen System selber nachprogrammieren.

Natürlich kann man sich dazu zwingen, permanent für mehrere Betriebssysteme gleichzeitig zu entwickeln. Der Programmierer ist dann gezwungen, sich auf das zu beschränken, was beide Betriebssysteme gemeinsam haben. Wer etwa für DOS/Windows und MacIntosh entwickelt, darf nur beschränkt Multitasking-Eigenschaften einsetzen. Er darf den Lautsprecher nur zum Piepsen verwenden. Er muss die Pixel-Auflösungen, Anzahl Mausknöpfe, Farbfähigkeit sämtlicher Zielsysteme berücksichtigen und in jedem Fall die schlechtere beider Welten wählen. Wenn er eine Anwendung gebaut hat, die er auf beiden Systemen zur Ausführung bringen kann, vergisst er ob seiner Begeisterung über die unermesslichen Vorteile der Portabilität, dass ihn das portable Programm dreimal so viel Aufwand gekostet hat, wie die separate Entwicklung in der angemessenen Sprache auf jeder der beiden Maschinen.

Für den Programmierer ist 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.

Die Unsitte grösserer Kunden (Banken, Behörden, etc.) uns auf untaugliche Instrumente festzulegen, kostet volkswirtschaftlich Milliarden.

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 ersten Mal 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. Dank der Verfügbarkeit von komfortablen Entwicklungswerkzeugen kann dieses Wissen in nützlicher Zeit in anderen Umgebungen wieder verwertet werden. 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 auch gelehrt, dass es äusserst gefährlich ist, „strategisch" zu denken. Niemand kann sagen, welche Fähigkeiten, welche Werkzeuge und welche Produkte wir Programmierer in fünf Jahren einsetzen müssen. Vor drei Jahren hätte niemand (naja: kaum jemand) die heutige Windows-Welle prognostiziert.

Produktivität und Kosten

Diejenigen Faktoren, die bei der Programmentwicklung bedeutend wesentlicher ins Gewicht fallen, als die Wahl der Programmiersprache, sind die Entwicklungshardware, die Compilers und die Entwicklerwerkzeuge.

Wenn man sich heute für *Borland C++* statt für *C* (oder für *Turbo Pascal 6.0* statt für *„Standard Pascal"*) entscheidet, hat man sich zwar von einer simplistisch verstandenen „Portabilität" abgekoppelt, was man dafür an Produktivität für das Zielsystem DOS/Windows gewonnen hat, ist nicht in Gold aufzuwiegen.

Die Entscheidung zur Verwendung von Objektbibliotheken für Windows bringt eine erhöhte Abhängigkeit vom Compilerlieferanten mit sich. Tatsächlich hat man sich schon in dem Moment in diese Abhängigkeit begeben, wo man entschied, überhaupt für Windows zu entwickeln. Das nicht-standardisierte Windows-Interface schliesst schon eine problemlose Portierung auf den MacIntosh aus, selbst wenn man nur ANSI-C benützt.

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 Ausbildungstand und die Vorlieben der Programmierer anpassen, wie man auch jedem anderen Handwerker die Freiheit zugesteht, seine Arbeitswerkzeuge selber zu bestimmen.

Zusammenfassung

Bei der Entscheidung für eine Entwicklungsumgebung prüfe man die realen Implementationen, die für das gewählte Betriebssystem, die Hardware und die Peripheriegeräte erhältlich sind und ihre Zusatzwerkzeuge. Man lege jedes Vorurteil bezüglich der einen oder anderen gut eingeführten Hochsprache beiseite und schätze den Lernaufwand für die verfügbaren Programmierer ab,

bei der Wahl des einen oder anderen Systems. Man überzeuge sich von der Verlässlichkeit des Lieferanten der Entwicklungsumgebung und schere sich einen Deut um sogenannte Standards.

Da man nichts dabei verlieren, aber einiges dabei gewinnen kann, wähle man eine objekt-orientierte Erweiterung der Sprache, welche ja den alten Standard als Subset unterstützt. Man lasse sich nicht von verqueren Portabilitätsüberlegungen davon abhalten, diese objekt-orientierten Eigenschaften auch zu nutzen. Wo die Programmierer auf *Pascal* eingefuchst sind, wähle man *Turbo Pascal 6.0* und *Turbo Pascal für Windows*, wo sie nur *C* können, entscheide man sich für *Borland C++* und die Object Windows Library. Wo die Fähigkeiten für *C* und *Pascal* etwa gleich verteilt sind, wird sich das Programmierteam wohl für *Pascal* entscheiden, weil diese Sprache dank ihrer formalen Spezifikation immer noch etwas schneller kompiliert werden kann.

Schliesslich laufe man nicht mit Scheuklappen vor Neuerungen davon. Das Erlernen der zweiten Programmiersprache mag für einen frischgebackenen Informatiker noch furchterregend sein. Die heutigen PC-Teenager haben mit Achtzehn sowieso schon im Mittel vier Sprachen hinter sich. Das Erlernen einer weiteren lohnt sich alleweil, wenn die Produktivität damit gesteigert werden kann. Die konkrete Implementation und die vorhandenen Utilities und Halbfabrikate sind dabei allerdings wichtiger als die Programmiersprache. *C* ist nicht gleich *C* und *Pascal* nicht gleich *Pascal*!

Juli 1992 Hartwig Thomas