

Make and Buy statt Make or Buy

Das Ziel, den Entwicklungsprozeß zu beschleunigen und Anwendungen ohne Qualitätseinbuße schneller zum Anwender zu bringen, ist nicht neu. Aber nach wie vor klaffen Anspruch und Wirklichkeit auseinander.

Die objektorientierte Software-Entwicklung ist nach Auffassung nicht weniger Experten gescheitert. Das Kernargument der Enttäuschten lautet: Wiederverwendung funktioniert nicht. Diesem Argument kann man sich je nach eigenen Erfahrungen anschließen oder es ebenso wiederlegen. Wiederverwendbarkeit ist durchaus keine Frage der Technologie, sondern ausschließlich der Organisation. Es gibt schließlich Unternehmen, die selbst in Zeiten prozeduraler Programmiersprachen Wiederverwendung durch sauber strukturierte Modulbaukästen tatsächlich realisierten.

Nachdem die Frage der Wiederverwendbarkeit in vielen Unternehmen noch nicht geklärt ist, eröffnen sich mit Komponententechnologien schon wieder neue Möglichkeiten. Dabei handelt es sich um Wiederverwendbarkeit auf höherer Ebene. "Alles was bereits auf dem Komponentenmarkt verfügbar ist, kaufen bzw. lizenzieren - alles was wirklich unternehmensspezifisch ist selbst entwickeln", lautet der Wahlspruch. Das Resultat ist eine Make and Buy-Strategie.

Sind die Unternehmen auf eine Make and Buy-Strategie vorbereitet? In aller Regel ist dies noch nicht der Fall. Haupthindernis ist noch immer die häufig projektzentrierte Organisation. Zwar können Rationalisierungsmöglichkeiten durch Lizenzierung von Software-Komponenten von externen Herstellern erschlossen werden, jedoch wird das Potential bei weitem nicht ausgeschöpft. Zum einen ist es kaum möglich, die Grundlage für eine technische Infrastruktur durch Entwicklung von Infrastruktur-Komponenten zu schaffen, zum anderen wird immer noch viel Aufwand in eigentlich überflüssige Parallelaktivitäten investiert.

Dabei eröffnet das Idealszenario verlockende Perspektiven. Das Unternehmen "stöpselt" Anwendungen aus vorhandenen, von Herstellern lizenzierte und selbst entwickelte Komponenten, zusammen. Sämtliche Komponenten sind bereits vollständig getestet und verfügen über definierte Schnittstellen. Sie lassen sich in unterschiedlichem Kontext einsetzen, so beispielsweise einmal als Element einer transaktionsorientierten, ein andermal als Element einer nicht transaktionsorientierten Anwendung. Wenn festgestellt wird, daß eine bestimmte Funktionalität noch nicht abgedeckt wird, ist die Frage zu stellen, ob diese auch in anderen Anwendungssystemen benötigt wird und somit mit einer Komponente abgedeckt werden könnte. Anschließend wäre zu klären, ob die gewünschte Komponente auf dem Komponentenmarkt verfügbar ist oder doch selbst entwickelt werden muß. Die aktive Beobachtung des Komponentenmarkts wird zu einer sinnvollen Aufgabe.

Die gegenwärtigen Entwicklungen im Software-Sektor gestatten es, auf die Erfüllung lange gehegter Erwartungen zu hoffen. Die Plattformen für verteilte Objekte (COM/DCOM-, CORBA- und Java-Plattformen sind etabliert. Die wichtigsten Dienste sind spezifiziert, wenn auch nicht immer ohne Schwächen. Gleichwohl ist die Weiterentwicklung nicht abgeschlossen. Die Entwickler werden sich auch künftig auch Änderungen der APIs einstellen müssen. Auf die Plattformen für verteilte Objekte wurden Komponentenmodelle gegründet, ActiveX (COM/DCOM) sowie JavaBeans und Enterprise JavaBeans (EJB, Java-Plattform). Das Feld wird durch ein Komponentenmodell für die CORBA-Plattform vervollständigt, das jedoch noch nicht endgültig spezifiziert ist. Nachdem einige Zeit im Raum stand, das EJB-Modell unter dem Label "CORBA Components" weitmöglichst zu übernehmen, steht dies mittlerweile offensichtlich nicht mehr zur Debatte. Das CORBA-Komponentenmodell wird jedoch einen Mapping-Mechanismus zum EJB-Modell spezifizieren.

Sämtliche Plattformen für verteilte Objekte wie auch die Komponentenmodelle verfolgen prinzipiell dieselben Ziele. Die technische Implementierung ist jedoch unterschiedlich gelöst und auch die APIs unterscheiden sich. Es ist nicht einfach, die so entstandenen Welten über Brücken miteinander zu verbinden. Die Plattformen für verteilte Objekte und die Komponentenmodelle bilden in ihrer Gesamtheit wiederum eine hervorragende Basis für Application Server als Ausführungsumfeld für Anwendungsobjekte und Komponenten ("paketierte Objektklassen"). Application Server übernehmen viele Infrastrukturaufgaben und entlasten damit die Anwendungsentwicklung ganz wesentlich.

In jüngerer Zeit wurden von Software-Herstellern eine Vielzahl von Application Server-Produkten entwickelt und im Markt eingeführt. Mit Komponententechnologien wird die bisherige Trennung in Web Server und Transaction Server hinfällig und die Produktlinien verschmelzen zum "Enterprise Application Server". Nach wie vor können Application Server für unterschiedliche Aufgaben eingesetzt werden, jedoch die technologische Basis

nähert sich immer mehr an. Web Server und Transaction Server bezeichnen nurmehr Rollen. Eine EJB kann sowohl auf einem Web Server als auch einem herkömmlichen Transaction Server ausgeführt werden.

Microsoft Transaction Server (MTS) implementierte als erster Application Server ein Komponentenmodell (COM/ActiveX). Sämtliche Wettbewerber setzen ausnahmslos auf das EJB-Komponentenmodell. Es wäre schließlich wenig sinnvoll, Microsoft auf ureigenem Terrain einen Wettbewerb zu liefern, zumal MTS quasi zum Nulltarif verfügbar ist. Insofern ist eine breite EJB-orientierte Allianz eine naheliegende Konsequenz. Von den derzeit mehr als 40 verfügbaren bzw. angekündigten Produkten mit EJB-Unterstützung werden sich mittel- und langfristig nur 4-5 Produkte im Verdrängungswettbewerb behaupten. Es werden die Produkte der "Großen" sein, die in mittelständischen und Großunternehmen den Fuß bereits mit anderen Produkten in der Tür haben.

Mit der hinreichenden Stabilität des CORBA-Komponentenmodells werden auch CORBA-basierende Application Server auf der Bildfläche erscheinen und das Bild vervollständigen. Es ist zu erwarten, jedoch dann in der Praxis zu beweisen, daß EJBs wirklich problemlos in CORBA-Container eingebettet werden können. Für die CORBA-orientierten Software-Hersteller ist die EJB-Unterstützung jedenfalls eine zwingende Notwendigkeit. Der "CORBA-Aufschwung" wurde schließlich nicht durch die plötzliche Erkenntnis der Fachwelt initiiert, daß CORBA eine wohl fundierte Technologie sei. Auslösend war vielmehr das in Kombination mit Java als Programmiersprache erkannte Potential.

Ohne jeden Zweifel wird sich die komponentenbasierte Anwendungsentwicklung in den nächsten Jahren durchsetzen. Die Frage stellt sich jedoch, ob die Software Engineering-Methoden und die CASE-Werkzeuge die komponentenbasierte Software-Entwicklung effizient unterstützen können. Ein Blick auf die gegenwärtige Methodenlandschaft macht noch eine ganze Reihe von Defiziten sichtbar.

Den Software-Engineering-Methoden liegt eine *Top-down*-Betrachtungsweise zugrunde. Der Einstieg erfolgt auf hoher Abstraktionsebene. Durch Verfeinerung, die durch formale Methoden unterstützt wird, wird das Problemfeld immer weiter exploriert und strukturiert bis am Ende die Anwendung konkrete Formen angenommen hat. Den vielen Vorteilen steht allerdings auch ein gravierender Nachteil gegenüber. Es wird nicht gefragt, ob Verallgemeinerungen möglich sind und Funktionen bzw. Objekte in identischer oder hochgradig ähnlicher Form auch an anderer Stelle (in anderen Anwendungen) vorkommen. Generalisierung allein, z. B. durch Abstrahieren von Klassen und das Bilden von Vererbungsbeziehungen, löst das Problem nicht.

Da der Software-Designer in der Regel über den Tellerrand des aktuellen Projekts hinaus blickt und auch andere Anwendungssysteme kennt, kann er natürlich sehr wohl erkennen, ob Funktionen bzw. Objekte auch anwendungsübergreifend nutzbar sind. Dies wird immer der Fall sein, wenn Dienste (z. B. Protokollierung von Fehlermeldungen) von mehreren Anwendungen in gleicher Weise genutzt werden können. Der Gedanke, Dienste in gemeinsam nutzbaren Komponenten zu verkapseln, liegt nahe.

Die *Top-Down*- muß durch eine *Bottom-up*-Betrachtungsweise ergänzt werden, um eine weitestmögliche Wiederverwendbarkeit von Software zu ermöglichen. Wird der Wiederverwendbarkeit von Software, wie zweifellos sinnvoll und wirtschaftlich zu begründen, eine hohe Priorität zugeordnet, müssen beide Betrachtungsweisen gleichberechtigt nebeneinanderstehen.

Die derzeit verfügbaren CASE-Werkzeuge unterstützen die *Bottom-up*-Vorgehensweise nicht. Eine Methodologie, wie potentielle Komponenten erkannt und anschließend modelliert werden, fehlt derzeit. Deshalb bleibt es dem Anwenderunternehmen nicht erspart, selbst ein Vorgehensmodell zu erarbeiten und dieses in einen oder mehrere Software-Entwicklungsprozesse abzubilden. Auch die Integration mit der vorhandenen CASE-Umgebung muß selbst gelöst werden. Diese Aufgabe ist nicht einfach zu realisieren.

Komponenten müssen bereits während der Analyse modellierbar sein. Jedes Unternehmen wird im Zeitverlauf ein Komponentenarsenal entwickeln und/oder von Software-Herstellern lizenzieren. Der globale Komponentenbestand muß abfragbar sein, damit Analytiker und Designer potentiell wiederverwendbare Komponenten auffinden können. Nach Erstellung und Abstimmung der Use Cases mit den Anwendern kann die textuelle Use Case-Beschreibung dazu genutzt werden, Objektklassen zu finden. Gleichzeitig sollten Analytiker und Designer jedoch auch in der Lage sein, den Komponentenbestand zu durchsuchen, um erkennen zu können, ob eine Klasse bereits durch eine Komponente abgedeckt wird. Dies zu erkennen, ist allerdings nicht einfach. Einerseits befindet sich das Klassen-Design noch im Fluß und andererseits umfaßt eine Komponente häufig mehrere Klassen. Auch unterschiedliche Namensgebungen können die Zuordnung sehr erschweren. Darüber hinaus müssen schon zu diesem Zeitpunkt technische Randbedingungen berücksichtigt werden, da jede Komponente auf einem bestimmten Komponentenmodell basiert. Möglicherweise kann eine bereits vorhandene Komponente doch nicht einge-

setzt werden, da das zugrundeliegende Komponentenmodell für das bestimmte Projekt nicht verwendet werden soll. Wenn beispielsweise eine ActiveX-Komponente eine bestimmte Funktionalität abdeckt, der Container jedoch nur EJB-Komponenten ausführen kann, ist die Komponente nicht nutzbar. Ist eine Komponente als Kandidat auserkoren, müssen ihre Schnittstellen in das Modell eingebracht werden können.

Die meisten CASE-Werkzeuge unterstützen mittlerweile UML als standardisierte Modellsemantik und Notation. Mit UML lassen sich durchaus auch Komponenten-Diagramme erstellen, obgleich dieser Diagrammtyp derzeit nur bei den wenigsten CASE-Werkzeugen implementiert ist. Es bedarf jedoch eines sorgfältig erarbeiteten Vorgehensmodells, um den Prozeß des Software Engineering zu strukturieren und die Qualität des entstehenden Produkts sicherzustellen. Das Vorgehensmodell definiert die Aktivitätentypen, die bei der Software-Entwicklung ausgeführt werden müssen und die Produkttypen, die dabei erzeugt werden. Es beschreibt, in welcher Folge und unter welchen Bedingungen Produkttypen entstehen. Ferner beschreibt das Vorgehensmodell die Rollen aller an der Software-Entwicklung beteiligten "Personentypen" und ordnet diese den Aktivitätentypen zu. Die meisten Vorgehensmodelle sind stark an den Möglichkeiten der eingesetzten CASE-Umgebung orientiert und stehen stets in enger Beziehung zu den angewandten Software Engineering-Methoden, der Methodologie. Andere Unternehmen übernahmen Prozesse von CASE-Herstellern oder wählten gar einen weitgehend neutralen Ansatz (insbesondere repräsentiert durch das V-Modell).

In der Gesamtschau betrachtet, unterstützen die Application Server-Produkte die komponentenbasierte Anwendungsentwicklung auf der technischen Ebene bereits jetzt ganz hervorragend. Die bekannten Vorgehensmodelle behandeln das Finden, Abgleichen und Modellieren von Komponenten jedoch stiefmütterlich, wenn überhaupt. Auch die CASE-Werkzeuge bieten nur eine rudimentäre und punktuelle Unterstützung. Darüber hinaus sind Erweiterungen der UML erforderlich, um die Semantik von Komponenten besser beschreiben zu können. Hier sind Analytiker und Designer derzeit noch auf sich selbst gestellt und müssen sich auf andere Weise, z. B. mit XML-Beschreibungen behelfen.

Die meisten der verfügbaren Server-Frameworks, wie beispielsweise die SanFrancisco-Frameworks der IBM, passen in ihrer bisherigen Form nicht mehr in die Landschaft. Sie realisieren zwar eine Komponentenstruktur, basieren jedoch nicht auf einem der zwei (künftig drei) wichtigsten Komponentenmodelle. Konsequenterweise trafen viele Hersteller bereits die Entscheidung, ihre Frameworks entsprechend anzupassen. So sind, um im Beispiel zu bleiben, künftig Komponenten des SanFrancisco-Frameworks als Enterprise JavaBeans-Komponenten realisiert und können von einem Application Server (WebSphere Application Server) ausgeführt werden.

Wenn die Software-Hersteller ihre Anwendungs-Frameworks (umfassen Anwendungsgrundgerüste und Infrastruktur) auf die Komponentenmodelle abgestimmt haben, eröffnen sich für die Anwenderunternehmen weitere Produktivitätssteigerungspotentiale. Vorhandene Komponenten können mit eigenentwickelten prinzipiell problemlos zusammenwirken. Andererseits können jedoch Protokollfragen (z. B. RMI über IIOP) Probleme bereiten und die Freude deutlich dämpfen. Ganz so heil wird die Welt deshalb nicht sein. Auch für die Hersteller von CASE-Werkzeugen bleibt noch viel Arbeit zu erledigen.

Heutige Make and Buy-Strategien sind noch zu sehr auf den Client konzentriert. Künftig wird es vor allem darum gehen, Server-Komponenten als "Plug-and-Play"-Komponenten lizenzieren zu können. Da Server-Komponenten im Gegensatz zu GUI-Komponenten nicht einfach vom Anwendungskontext losgelöst werden können, wird es keine "neutralen" Server-Komponenten geben. Server-Komponenten werden deshalb immer ein zugrundeliegendes Framework, etwa das SanFrancisco-Framework, voraussetzen. Mittel- und langfristig werden einige wenige Software-Hersteller den Markt der Anwendungs-Frameworks unter sich aufteilen. Anwenderunternehmen werden in der Lage sein, selbst Server-Komponenten zu entwickeln und in die Anwendungsinfrastruktur einzubetten. Dies bedeutet in jedem Fall einen bedeutenden Produktivitätsgewinn, der zwar nicht in einer allgemeinverbindlichen Zahl ausgedrückt werden kann, aber im allgemeinen bei über 20% liegen wird.