

## Komponentenbasierte Anwendungsentwicklung - ist ein Return on Investment erzielbar?

Autor: Dieter E. Jenz, Febr. 1999

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 widerlegen. 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 erfolgreich realisierten. Während die Frage der Wiederverwendbarkeit in vielen Unternehmen noch nicht geklärt ist, eröffnen sich mit Komponententechnologien neue Möglichkeiten.

Komponenten haben den Charakter "unabhängiger" Objekte. Sie können unabhängig voneinander entwickelt, in Anwendungen integriert und ohne Nebenwirkungen auf die Umgebung ersetzt werden. Sie müssen nicht notwendigerweise mit objektorientierten Programmiersprachen entwickelt werden. Prozedurale Programmiersprachen sind durchaus geeignet, jedoch eröffnen objektorientierte Programmiersprachen viele Vorteile. Über die Objekteigenschaft hinaus stellen Komponenten zusätzliche Anforderungen an Objektmodelle. Die Kontextabhängigkeiten von der Umgebung müssen in den Schnittstellen beschrieben werden. So muß z. B. in der Schnittstelle deklariert werden, ob die Komponente im Transaktionskontext ausgeführt wird. Schnittstellen können somit auch als Verträge (*Contracts*) zwischen Komponenten und Umgebung verstanden werden. In dieser Betrachtungsweise ist eine Komponente mehr als ein Objekt. Sie stellt eine Paketierung für Objektklassen bereit.

In der Gesamtschau betrachtet müssen Komponenten die folgenden Anforderungen erfüllen:

- Verbergen von Implementierungsdetails: Schnittstelle und Implementierung sind strikt voneinander getrennt. Der Anwendungsentwickler "sieht" lediglich die Schnittstelle;
- Eindeutige Identifizierbarkeit: Jede Komponente ist zu jeder Zeit eindeutig identifizierbar und unterscheidbar. Die Identifikation kann über eine Art "Vertrags-Nummer" geschehen;
- Beliebige Kombinierbarkeit: Komponenten können in beliebigen Kombinationen genutzt werden. Die Art der Nutzung und die Funktionalität einer Komponente sind voneinander unabhängig. So kann z. B. eine Adreßverwaltungskomponente in vielen Anwendungen genutzt werden.

Komponenten sind loser gekoppelt als Objekte, die aufgrund der objektorientierten Prinzipien eine ganze Reihe relativ enger Abhängigkeiten verkörpern. Genannt sei vor allem die Vererbung der Implementierung (implementation inheritance). Objektorientierte Anwendungssysteme bestehen aus relativ feinkörnigen Objekten mit starken Schnittstellenabhängigkeiten. Komponenten sind grobkörniger als Objekte (mit nur wenigen Ausnahmen). Sie unterstützen prinzipiell mehrere Schnittstellen und ermöglichen dadurch Flexibilität. Dies zeigt sich insbesondere bei der Weiterentwicklung von Komponenten. Bereits vorhandene Schnittstellen können weiterhin unterstützt werden, wodurch bereits vorhandene Anwendungen nicht geändert werden müssen. Neue Anwendungen können neue Schnittstellen nutzen. Daß "alte" und "neue" Anwendungen unterschiedliche Schnittstellen nutzen und koexistieren können, ist ein enormer Vorteil.

Ein weiterer gravierender Unterschied zwischen Objekten und Komponenten besteht in der Statusverwaltung. Objekte verwalten ihren Status in den Objektattributen. Solange das Objekt aktiv ist, kann jede aufgerufene Methode auf die gerade aktuellen Werte der Objektattribute zugreifen. Komponenten sind im Gegensatz zu Objekten statuslos, d. h. mit jedem Aufruf einer Komponente enthalten die Komponentenattribute den Initialzustand. Es ist somit nicht möglich, die Attributwerte über mehrere Komponentenaufrufe hinweg zu "retten". Andererseits ist dies auch nicht erforderlich.

Schon allein aus den oberflächlich dargestellten technischen Gründen muß die Frage der Wiederverwendbarkeit und des erzielbaren Nutzens wieder neu aufgeworfen werden. Selbstverständlich muß zunächst auch aus wirtschaftlicher Sicht die Frage beantwortet werden, ob Reuse bei dynamischem technologischem Fortschritt überhaupt realisierbar ist. Wenn die bekannte Faustregel zugrundegelegt wird, gemäß der der Aufwand für die Erstellung einer wiederverwendbaren Komponente um das 5-7fache höher ist (im Einzelfall kann der Faktor höher oder niedriger liegen), muß demgemäß eine Komponente mindestens 5 mal wiederverwendet, also insgesamt 6 mal eingesetzt werden, um einen positiven Return on Investment zu erzielen. Die Hürde scheint sehr hoch, gemessen an den bisherigen Erfahrungen. Bereits der schnelle technologische Fortschritt verhinderte in der Tat eine nennenswerte Wiederverwendung vorhandener Bausteine für die Software-Entwicklung.

Eine nüchterne Betrachtung des technologischen Fortschritts zeigt ein Abflachen der Kurve. Die Plattformen für verteilte Anwendungen (COM/DCOM-, CORBA- und Java-Plattform) haben sich etabliert und eine hinreichende Stabilität erreicht. Auch die Komponenten-Modelle (COM/ActiveX-, JavaBeans- und Enterprise JavaBeans-Modelle) sind etabliert. Das CORBA-Komponentenmodell ist zumindest in Form eines Spezifikationsentwurfs umrissen. Es besteht Klarheit, welche Funktionalität von einer Plattform für verteilte Anwendungen wie auch einer Komponenten-Plattform gefordert ist. Sämtliche aktuellen Entwicklungen haben zum Ziel, noch fehlende Funktionalität zu implementieren. In der Gesamtschau kann festgestellt werden, daß heute nahezu alle Dienste, Schnittstellen und Protokolle spezifiziert sind.

Alle Anzeichen weisen darauf hin, daß Schnittstellen und Protokolle sich in Zukunft nur wenig ändern werden. Auch auf der Seite der Software Engineering-Methoden und Beschreibungssprachen haben sich mit UML (Unified Modeling Language) und OCL (Object Constraint Language) Standards durchgesetzt, die - zumindest was UML anbelangt - mittlerweile von nahezu allen CASE-Werkzeugen unterstützt werden. Schließlich haben sich auch bei den Programmiersprachen die Fronten geklärt. In der Gesamtschau betrachtet ist damit eine hinreichende Basis geschaffen, die bisherigen Vorbehalte neu zu bewerten und Wiederverwendbarkeit konkret zu planen. Planung ist deshalb unverzichtbar, da Wiederverwendbarkeit auch bei der Entwicklung von Komponenten kein Abfallprodukt ist. Ohne organisatorische Maßnahmen entstehen Komponenten, die sich in ihrer Funktionalität ähneln, jedoch nicht decken.

Die bisherige Faustregel (Faktor 5-7) gilt unter den veränderten Vorzeichen nicht mehr. Die Entwicklung wiederverwendbarer Komponenten verursacht nur noch den 2-4fachen Mehraufwand, wobei die Lernkurve bereits als durchlaufen vorausgesetzt wird. Dem gegenüber steht bei konventioneller Entwicklung ein "natürlicher" Mehraufwand, der sich durch redundante Analyse-, Design-, und Entwicklungsaktivitäten ergibt. Der Anteil der Infrastrukturfunktionen an einer typischen Anwendung beträgt etwa 50-70%, wobei dieser Anteil bei verteilten Anwendungen aufgrund höherer Komplexität noch zunimmt. Die Schere zwischen komponentenbasierter und konventioneller Anwendungsentwicklung wird somit deutlich kleiner. Unter diesen Voraussetzungen kann es nur sinnvoll sein, die komponentenbasierte Anwendungsentwicklung konsequent anzugehen. In der Konsequenz muß in der IT-Abteilung eine "Wiederverwendungskultur" entstehen. Organisatorische Maßnahmen sind jedoch zwingend erforderlich und entscheiden letztlich über Erfolg und Mißerfolg der komponentenbasierten Anwendungsentwicklung.