

Unterstützung der komponentenbasierten Anwendungsentwicklung durch Metriken

Autor: Dieter E. Jenz

In der heutigen Wettbewerbssituation wird vom Management gefordert, Anwendungssysteme schneller in den operationalen Betrieb zu bringen. Dazu ist eine maßgebliche Produktivitätssteigerung unerlässlich. Auch wenn moderne Entwicklungsumgebungen die Produktivität in begrenztem Rahmen steigern helfen, werden Produktivitätszuwächse sofort wieder durch die höhere Komplexität verteilter Anwendungen zunichte gemacht und sogar überkompensiert. Produktivitätszuwächse können deshalb mit den Anforderungen nicht Schritt halten.

Nicht erst seit heute wird ein wichtiger Beitrag zur Lösung des Produktivitätsproblems in der komponentenbasierten Anwendungsentwicklung gesehen. Komponenten sind unabhängig von Anwendungen und können prinzipiell von beliebig vielen Anwendungen genutzt werden. Sie können selbst entwickelt oder von Software-Herstellern lizenziert werden. Anwendungen stellen in diesem Sinn einen Rahmen zur Verfügung, innerhalb dessen Komponenten miteinander und auch mit der "Außenwelt" zusammenwirken können. Um das koordinierte Zusammenwirken zu ermöglichen, enthält die Anwendung "Klebstoff" in Form von Programm-Code.

Die komponentenbasierte Anwendungsentwicklung muß gezielt und mit voller Unterstützung des Unternehmens- und des IT-Managements eingeführt werden. Es handelt sich um eine strategische Richtungsentscheidung, die mit vielerlei Konsequenzen verbunden ist. Erst nachdem die notwendigen Entscheidungen getroffen und der organisatorische Rahmen definiert ist, kann überhaupt an die komponentenbasierte Anwendungsentwicklung herangegangen werden. Dann erst sind Gedanken darüber, wie Komponenten zu "schneiden" sind, sinnvoll.

Ausgehend von der prinzipiell anwendungsunabhängigen Natur von Komponenten stellt sich die Frage, wie der funktionale Umfang einer Komponente bestimmt werden soll. Die allgemeinen Leitlinien lauten:

- eine Komponente deckt einen bestimmten isolierten fachlichen Teilaspekt ab;
- der funktionale Umfang sollte so gewählt werden, daß eine Komponente möglichst häufig wiederverwendet werden kann;
- es darf keine Abhängigkeiten zwischen Klassen geben, die sich in unterschiedlichen Komponenten befinden. Wenn sich dies nicht vermeiden läßt, sollten nur unidirektionale Beziehungen zugelassen werden. Andernfalls entstehen ungünstige Abhängigkeiten, d. h. Komponenten können nur gemeinsam wiederverwendet werden;
- eine technische Klasse darf nie eine fachliche Klasse benutzen. Umgekehrt ist dies jedoch selbstverständlich zulässig.

Tendenziell entstehen kleine Funktionseinheiten, wenn das Wiederverwendbarkeitsprinzip zur Maxime erhoben wird. Im Hintergrund steht das Prinzip: eine Schnittstelle pro Funktionseinheit. Wenn jedoch Objekte und damit auch Komponenten über mehrere Schnittstellen angesprochen werden können, gilt das Prinzip des kleinsten gemeinsamen Nenners nicht mehr. Jedem Client (Aufrufer einer Funktion bzw. Übermittler eines Methodenaufrufs) kann im Prinzip eine individuelle Schnittstelle zugeordnet werden, die einer individuellen Sicht auf die Komponente entspricht. Später können weitere Schnittstellen hinzugefügt werden, ohne notwendigerweise in die Implementierung, den Programm-Code, eingreifen zu müssen. Obwohl der kleinste gemeinsame funktionale Nenner nicht mehr Maßstab für die Komponentengröße ist, ist damit kein Freibrief für "riesige" Komponenten ausgestellt.

Die Plattformen für verteilte Anwendungen unterstützen multiple Schnittstellen in unterschiedlicher Weise. Während etwa bei COM/DCOM multiple Schnittstellen von vornherein unterstützt werden, kam diese Möglichkeit bei CORBA erst im vergangenen Jahr hinzu. Nur wenige CORBA-Implementierungen unterstützen derzeit bereits multiple Schnittstellen.

Unter wirtschaftlichen Aspekten stellt sich die Frage, wie häufig eine Komponente eingesetzt werden kann und welcher Mehraufwand für die Komponentenerstellung anfällt. Als Faustregel kann Faktor 2-4 gelten, wobei im Einzelfall Faktor 4 auch deutlich übertroffen werden kann. Dennoch zeigt sich, daß der "Break-even-Point" dank moderner Technologien niedriger liegt als früher. Unter wirtschaftlichen Gesichtspunkten ist es somit in jedem Fall sinnvoll, auf die komponentenbasierte Anwendungsentwicklung zu setzen. Die in der Vergangenheit gewonnenen Erfahrungen können vor dem Hintergrund neuer Technologien (Komponenten-Plattformen) nicht länger als Maßstab gelten. Auch wenn gezielt und konkret für Wiederverwendbarkeit entworfen wurde, konnte vielfach der gewünschte Wiederverwendungseffekt nicht erzielt werden, da sich die technologische Basis wie auch die fachlichen Anforderungen änderten. Zumindest die technologische Basis ist heute weitgehend stabil. Die Ände-

rungen fachlicher Anforderungen lassen sich durch "eingebaute" Konfigurierbarkeit von Komponenten wesentlich besser abfedern.

Den bisherigen Erfahrungen zufolge steht dem höheren Aufwand für die Entwicklung wiederverwendbarer Komponenten eine höhere Qualität gegenüber. Wie Untersuchungen zeigen (Matsumoto, Masao, "Research to Shed Light on Software Shadows," IEEE Software, Vol. 12, No. 5, September 1995, S. 16), kann die Qualität wiederverwendbarer Software (Klassen-Bibliotheken, Frameworks) bis zu 10 mal höher sein als die Qualität einer nicht für Wiederverwendung vorgesehenen Software. Diese Aussage kann auch ohne Einschränkungen auf Komponenten übertragen werden. Dies bedeutet, daß der Wiederverwender einer Komponente von geringeren Kosten im Lebenszyklus ausgehen kann, die mit der Anzahl wiederverwendeter Komponenten steigen. Darüber hinaus wird er auch nicht mit Kosten belastet, falls Fehler in Komponenten gefunden werden und beseitigt werden müssen. Der "Aufwandsfaktor" (Faktor 2-4, s.o.) muß deshalb relativiert und im Zusammenhang deutlicher Kosteneinsparpotentiale gesehen werden.

Qualität und Effizienz der Software-Entwicklung werden schon seit langem mit Hilfe von Metriken bewertet. Im Projektverlauf werden in regelmäßigen Abständen Zustandsdaten erhoben und, sofern möglich, mit Sollwerten in Beziehung gesetzt. Metriken existieren für die Entwicklung prozeduraler und objektorientierter Systeme. Auch die komponentenbasierte Anwendungsentwicklung muß mit Metriken unterstützt werden, um aussagekräftige Daten über Qualität, Reife und Effizienz zu gewinnen. Auf Basis der gewonnenen Zahlen können gezielte Verbesserungen im Software-Entwicklungsprozeß angesetzt werden. Gleichzeitig sind auch Aussagen über das aktuelle Projektrisiko im Hinblick auf Ressourcenmehraufwand und mögliche Terminverzögerungen möglich.

Die für objektorientierte Anwendungssysteme genutzten Metriken gelten selbstverständlich auch für komponentenbasierte Anwendungssysteme (sofern es sich nicht um "prozedurale" Komponenten handelt). Darüber hinaus sind weitere Metriken sinnvoll. Um Qualität und Effizienz der komponentenbasierten Anwendungsentwicklung beurteilen zu können, sollte möglichst frühzeitig ein Rahmenwerk von Metriken entwickelt werden. Bei der konkreten Anwendung muß auch der Projektumfang berücksichtigt werden. Deshalb werden den einzelnen Projektkategorien (kleine, mittlere, große Projekte) im Rahmen eines Tailoring die anzuwendenden Metriken zugeordnet.

Das Rahmenwerk gliedert in globale Metriken, die den Fortschritt der komponentenbasierten Entwicklung projektübergreifend meßbar machen. Globale Metriken sind relativ unscharf, da oft keine Vergleichsgrößen herangezogen werden können. Zu den globalen Metriken zählen:

- Anzahl der eingesetzten Komponenten. Es ist nicht möglich, die Anzahl der eingesetzten Komponenten in Beziehung zu setzen, da nicht bekannt ist, wieviele Komponenten im Optimalfall genutzt werden würden. Auch der Weg über die Anzahl der Klassen ist nicht hilfreich, da bei lizenzierten Komponenten nicht bekannt ist, wieviele Klassen diese umfassen. Immerhin ist jedoch im Vergleich von (Teil-)Projekten erkennbar, ob einzelne Projekte unverhältnismäßig wenig Komponenten nutzen und damit Anlaß für weitere Untersuchungen gegeben ist;
- Anzahl neu entwickelter Komponenten. Auch hier fehlt eine objektive Bezugsgröße. Die ersten Projekte werden relativ viele neu entwickelte Komponenten hervorbringen. Im Zeitverlauf nimmt die Anzahl neu entwickelter Komponenten kontinuierlich ab.
- Anzahl überarbeiteter Komponenten. Bezugspunkt sind die Implementierungen, nicht die Schnittstellen. Dabei wird davon ausgegangen, daß Schnittstellen nachträglich nicht mehr verändert werden (immutable interfaces). Die Anzahl der Überarbeitungen wird mit der Gesamtzahl der eingesetzten selbst entwickelten Komponenten in Beziehung gesetzt. Je ungünstiger das Verhältnis, desto höher ist die Wahrscheinlichkeit eines chronisch unausgereiften Komponentendesigns.
- Stabilität der Architektur. Komponenten werden auf Basis einer sorgfältig erarbeiteten Architektur entwickelt, die für sämtliche Komponenten gilt. Die Stabilität der Architektur bemißt sich nach der Zahl der erforderlichen Eingriffe.

Wie bereits angedeutet, wächst das Komponentenportfolio erst im Zeitverlauf. Die ersten Projekte können fast ausschließlich nur lizenzierte Komponenten nutzen. Deshalb muß der Komponenteneinsatz stets im Zusammenhang mit dem aktuell vorhandenen Komponentenportfolio beurteilt werden.

Es ist prinzipiell sinnvoll, in Client- und Server-Komponenten zu untergliedern. Client-Komponenten werden sehr viel häufiger eingesetzt als Server-Komponenten. Das Angebot an Client-Komponenten besteht zum großen Teil aus GUI-Komponenten (z. B. zur Darstellung von Daten in spezifischer tabellarischer Form, Fortschrittsanzeigen u. v. m.) und ist bereits sehr umfangreich. Die meisten Client-Komponenten werden direkt oder indirekt (über die Entwicklungswerkzeuge) von Software-Herstellern lizenziert und nicht selbst entwickelt. Gerade

Server-Komponenten bringen jedoch quantitativ betrachtet ungleich höhere Entlastungseffekte bei Wiederverwendung. Die meisten Server-Komponenten müssen selbst entwickelt werden, es sei denn es entwickelt sich ein branchenspezifischer Komponentenmarkt. Dafür ist jedoch eine definierte "Branchen-Anwendungsarchitektur" unabdingbare Voraussetzung.

Die globalen Metriken können um komponentenspezifische Meßgrößen ergänzt werden. Für jede Komponente sind die folgenden Informationen hilfreich:

- Anzahl der Wiederverwendungen: der Wert sagt aus, wie häufig eine Komponente unmodifiziert wiederverwendet wurde. Bezugspunkt ist die Implementierung, nicht die Schnittstelle(n). Die Anzahl der Wiederverwendungen ist als Meßgröße sowohl für selbst entwickelte als auch lizenzierte Komponenten sinnvoll.
- Ressourceneinsatz: sagt aus, wieviele Personenstunden in Entwicklung und Pflege einer Komponente investiert wurden.
- Kosten: sagt aus, welchen finanziellen Aufwand Entwicklung und Pflege einer Komponente zu internen Verrechnungspreisen verursacht haben.

Wie bereits erwähnt, sind die üblichen, komponentenunabhängigen, Metriken nach wie vor sinnvoll anwendbar. Dazu zählen insbesondere:

- Stabilität des Design: hält fest, wieviele Designänderungen pro Iterationsstufe vorgekommen sind. Aus dem Kurvenverlauf läßt sich erkennen, welchen Stabilitätsgrad das Design erreicht hat.
- Funktionalität: ermittelt auf Grundlage von definierten Testfällen im Rahmen eines Black box-Tests, zu welchem Grad geforderte Funktionalität bereits erfolgreich nachgewiesen wurde. Untergliedert werden kann in fachliche und technische Testfälle. Meßbar wird sowohl die Testabdeckung als auch der Testerfolg.
- Integrität des Design: ermittelt im Rahmen eines White box-Tests Informationen über die Integrität des Software-Design. Auch hier wird die Testabdeckung mit dem Testerfolg in Beziehung gesetzt. Allerdings ist es schwierig, überhaupt die Anzahl der Testfälle zu ermitteln.
- Fehlerbeseitigungsstatus: Hier werden Meßgrößen wie z. B. die Anzahl der entdeckten und beseitigten Fehler, das durchschnittliche Alter der noch nicht beseitigten Fehler sowie der durchschnittliche Zeitbedarf für die Fehlerbeseitigung festgehalten.

Die Metriken sollten in regelmäßigen Zeitabständen erhoben werden. Sinnvoll sind monatliche Abstände. Die erhobenen Daten können dann in der Zeitachse aufgetragen und grafisch dargestellt werden. Bereits mit wenigen einfachen Beobachtungsgrößen läßt sich der Erfolg der komponentenbasierten Anwendungsentwicklung erkennen. Nach einiger Zeit, wenn sich die komponentenbasierte Anwendungsentwicklung im Unternehmen etabliert hat und einiges Datenmaterial vorliegt, können auch Return-on-Investment-Betrachtungen angestellt werden.

Im Mittelpunkt der komponentenbasierten Anwendungsentwicklung steht vor allem der wirtschaftliche Aspekt. Über die Zeit hinweg kann mit den gewonnenen Zahlen eine Aussage darüber getroffen werden, ob der Komponenteneinsatz effizient geschieht. Nach den bisherigen Erfahrungen ist nicht von der Hand zu weisen, daß vorhandene Komponenten manchmal nur widerwillig genutzt werden und Entwickler nach dem Motto "not invented here" stets zu einer Neuentwicklung tendieren. Um den Erfolg der komponentenbasierten Anwendungsentwicklung zu fördern, ist deshalb eine laufende Beobachtung des Komponenteneinsatzes unverzichtbar.

Die Wiederverwendung von Komponenten ist nicht zum Nulltarif zu haben. Wiederverwendet werden kann nur, was auch wiedergefunden werden kann. Sämtliche Versuche, die Wiederverwendbarkeit zu fördern, müssen ohne eine entsprechende Infrastruktur ins Leere greifen. Gefordert sind in jedem Fall ein Komponenten-Repository und Werkzeuge, die das Auffinden von Komponenten nach diversen Suchkriterien ermöglichen. Ohne eine geeignete Infrastruktur und eine wiederverwendungsfördernde Organisation bleiben deshalb komponentenorientierte Metriken ohne jeden Sinn.