

Teilen und herrschen

Abstraktion ist eines der wichtigsten Werkzeuge von Softwareentwicklern. Nur durch **Abstraktion**, also die Möglichkeit Details ausblenden zu können, sind wir in der Lage, **komplexe Problemstellungen zu bewältigen**. Bei der Beurteilung, ob eine gewählte Abstraktion angemessen ist, helfen die beiden Prinzipien »Single Level of Abstraction« (SLA) und »Single Responsibility Principle« (SRP).

Schon die Römer wussten, wie man ein großes Land regiert. Sie teilten es in kleinere Teile auf und herrschten dann in diesen kleineren Einheiten. Waren auch diese zu groß, wurde weiter geteilt. Dies trieb man einfach so lange, bis handhabbare Einheiten entstanden waren. Das dahinterliegende Prinzip lautet: teile und herrsche.

Das Prinzip ist auch bei der Softwareentwicklung anzutreffen. Ja, ich behaupte sogar: Ohne Aufteilen eines Problems in Teilprobleme ist Softwareentwicklung schlicht nicht möglich. Größere Probleme in einem Happen verspeisen zu wollen, schlägt oft genug fehl. Beherrschen lassen sich die großen Happen erst, nachdem sie in kleinere Stückchen zerlegt sind.

Das Aufteilen eines Problems in Teilprobleme findet rekursiv statt. Zunächst wird das Gesamtproblem in Teile zerlegt. Diese Teile werden dann ebenfalls wieder zerlegt. So ergibt sich am Ende eine Hierarchie, eine Baumstruktur: Oben steht das Gesamtproblem in seiner vollen Komplexität, je weiter man im Baum nach unten schaut, desto geringer ist die Komplexität der einzelnen Teilprobleme. Dies setzt sich so lange fort, bis auf unterster Ebene die Komplexität der Teilprobleme so klein ist, dass man sie in kurzer Zeit lösen kann. Am Ende arbeiten die kleineren Teile natürlich wieder zusammen an einem übergeordneten größeren Ganzen.

Eine Hierarchie von Funktionseinheiten

Auch bei der Implementierung arbeiten wir mit hierarchischen Strukturen. Hier zerlegen wir nicht Probleme in Teilprobleme, sondern größere Funktionseinheiten in kleinere Funktionseinheiten. Dabei ist »Funktionseinheit« ein Überbegriff, der in die zur Verfügung stehenden Konstrukte der jeweils verwendeten Programmiersprache zu übersetzen ist. Im Falle der Objektorientierung haben wir es bei Funktionseinheiten zunächst einmal mit Methoden und Klassen zu tun. Eine Funktionseinheit ist also im allereinfachsten Fall eine Methode. Setzt man mehrere Methoden zusammen, ergibt sich eine Klasse. Setzt man mehrere Klassen zusammen, ergibt sich eine Assembly. Auch Assemblies können wieder zu einer größeren Funktionseinheit zusammengefasst werden, nämlich einer Komponente.

Eine Komponente unterscheidet sich von einer Assembly dadurch, dass sie einen expliziten und separaten Kontrakt hat. Dabei meint »explizit«, dass der Kontrakt sich eben nicht implizit per Reflection aus der Assembly ergibt, sondern explizit durch ein oder mehrere Interfaces beschrieben ist. Separat meint, dass dieser Kontrakt nicht in derjenigen Assembly enthalten ist, die die

Komponente implementiert, sondern davon getrennt in einer eigenen Assembly abgelegt ist.

Auch Komponenten können wieder zu größeren Einheiten zusammengefasst werden, nämlich zu einem Prozess. Auf dieser Ebene erst entsteht ein ausführbarer Betriebssystemprozess – oder vereinfacht: eine EXE-Datei. In vielen Fällen endet die Hierarchisierung bei einem Prozess. Dennoch können auch mehrere Prozesse wieder zu einer größeren Funktionseinheit zusammengefasst werden, wenn die Gesamtaufgabe zu komplex für einen einzelnen Prozess ist. Dies soll hier aber nicht weiter thematisiert werden.

Am Ende entsteht so also ein Baum von Funktionseinheiten. Ziel bei der Aufteilung in eine Hierarchie von Funktionseinheiten ist die Abstraktion. Auf jeder Ebene stellt eine Funktionseinheit eine definierte Funktionalität zur Verfügung. Wie sie ihre Aufgabe im Detail bewerkstelligt, muss auf dieser Ebene nicht interessieren. Von den Details wird abstrahiert. Erst wenn man im Baum tiefer nach unten geht, tauchen die Details einer Funktionseinheit auf. Auch sie sind jeweils für sich genommen wieder Einheiten mit einem definierten Funktionsumfang, zuständig für einen Teilaspekt der übergeordneten Funktionseinheit. So kann ich auf jeder Ebene entschieden, wie viele Details ich sehen will.

Abstraktion ist Filterung von Details

Am Beispiel einer Klasse stellt sich die Abstraktion wie folgt dar: Eine Klasse ist zuständig für einen klar umrissenen Funktionsumfang. Idealerweise sollte sich die Aufgabe der Klasse bereits aus ihrem Namen erschließen lassen. Bin ich nur an einem groben Überblick interessiert, genügt der Klassenname manchmal als Information aus. Erst wenn mich interessiert, wie genau die Klasse ihre Aufgabe löst, blende ich die Details ein, schaue ich in die Klasse hinein.

Innerhalb einer Klasse wird die Gesamtaufgabe zerlegt. Dabei kommen in der Regel Methoden zum Einsatz, in Einzelfällen auch geschachtelte Klassen. Ist die Teilaufgabe, für die eine einzelne Methode zuständig ist, zu umfangreich, um sie in dieser Methode bereits komplett zu bewerkstelligen, wird die Methode in weitere Methoden zerlegt. Dies setzt sich so lange fort, bis jede einzelne Methode hinreichend klein ist. Ob dies der Fall ist, kann man daran erkennen, dass es leicht

STEFAN LIESER

Blog:
lieser-online.de
E-Mail:
stefan@lieser-online.de



Stefan Lieser ist Software-Entwickler aus Leidenschaft. Sein Interesse gilt den agilen Methoden der Software-Entwicklung, und er sucht ständig nach Verbesserung und neuen Wegen. Bereits während des Informatikstudiums setzte er seinen Schwerpunkt im Bereich Softwaretechnik. Über die Jahre hat er sich intensiv mit Patterns und Principles auseinandergesetzt. Dies spiegelt sich in seinen zahlreichen Vorträgen wider. Ferner arbeitet er als Berater und Trainer im .NET-Umfeld. Gemeinsam mit Ralf Westphal hat er die *Clean Code Developer*-Initiative ins Leben gerufen.

BlogEngine.NET
Version 1.6



Die Entwickler des ASP.NET-Open-Source-Projekts „BlogEngine.NET“ haben kürzlich die Version 1.6 vorgestellt.

Als eine wesentliche Neuerung ist zur Vermeidung von Spam-Belästigung das neue Kommentarmanagement und die Spam-Filter-Infrastruktur hinzugekommen. Kommentare können über alle Postings hinweg gesichtet, bearbeitet, gelöscht und verschoben werden. Sie können weiterhin manuell moderiert werden, doch eine automatisierte Validierung durch lokale oder externe Dienste ist jetzt ebenfalls möglich. Eigene Spam-Filter können aufgrund der PlugIn-Struktur von BlogEngine.NET erstellt und eingebunden werden.

Ein weiteres neues Feature ist die Unterstützung von mehreren Widget-Zonen. Widgets können nun überall auf einer Seite platziert und zwischen Zonen verschoben werden.

Info: dotnetblogengine.net/page/BlogEngineNET-16-Release-Notes.aspx

Windows 7



400 Seiten
deutsch
erscheint 03/10
ISBN:
978-3-86645-539-9
EUR 29,90

Windows 7 für Entwickler Crashkurs

Laurence Moroney, Yochay Kiriaty,
Sasha Goldshtein
Microsoft Press

VB.NET

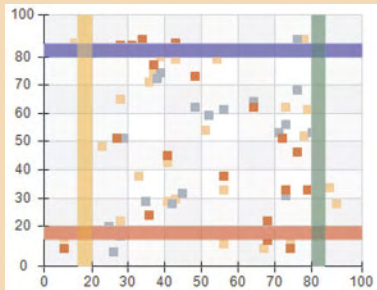


1000 Seiten
deutsch
erscheint 04/10
ISBN:
978-3-86645-535-1
EUR 49,90

Microsoft Visual Basic 2010 Das Entwicklerbuch

Klaus Löffelmann
Microsoft Press





Die erste Version dieses Jahres hat Nevron mit dem Release Nevron .NET Vision 2010 Vol.1 veröffentlicht. Unter vielen Verbesserungen und neuen Features sind beispielsweise Vektor-Export nach PDF, Flash, Silverlight (XAML) und EMF hinzugekommen. Zusammen mit dem bereits früher verfügbaren SVG-Export bietet die Nevron .NET Vision Suite nun eine umfassende Unterstützung sowohl für Rich Internat Anwendungen (RIA) als auch für Druckmedien im Rahmen einer einzigen, konsistenten API an. Damit lassen sich komplexe BI-Dashboards, SCADA-Interfaces, Workflows, Org-Charts und mehr erstellen, die heutigen Unternehmensanforderungen genügen. Alle Komponenten sind nunmehr vollständig lokalisierbar.

Die .NET-Charting-Komponente bietet jetzt auch eine eingebaute Unterstützung zur Visualisierung von umfangreichen Datenmengen an. Dies wird durch fortgeschrittene Multithread-Clustering- und Sampling-Algorithmen erreicht.

Das .NET Diagramming-Framework nutzt nun automatisch die Leistung von Multicore-Prozessoren, indem zum Rendern von komplexen Layouts mehrere Threads verwendet werden. Die Möglichkeit, WinForms-Controls einzubetten erlaubt das Gestalten aufwändiger gemischter Benutzeroberflächen.

Eine voll funktionsfähige Evaluierungsversion kann bei Nevron heruntergeladen werden.

Info: www.nevron.com

Anzeige

F#
Funktionale Programmierung mit Visual Studio 2010

15.–16.03.2010
Köln

22. – 23.03.2010
Karlsruhe
EUR 949,-

Info:
ppedv.de/schulung/kurse/FSharp-FunktionelleProgrammierung-Datenflussprogramme-Grundlagen.aspx

fällt, einen treffenden Namen für die Methode zu finden. Ferner kommt man beim Zerlegen irgendwann an den Punkt, an dem eine weitere Zerlegung nicht mehr möglich ist, da die Methode bereits für eine einzige, klar definierte Verantwortlichkeit zuständig ist.

Auch auf der Ebene der Methoden kann ich entscheiden, wie viele Details ich betrachten möchte. Bin ich nur an einem groben Überblick interessiert, schaue ich lediglich in die öffentlichen Methoden. In einfachen Fällen befinden sich darin bereits die wenigen Anweisungen, die nötig sind, um die Aufgabenstellung der Methode zu lösen. Oft wird die Aufgabe der Methode jedoch weiter zerlegt, es kommen nicht-öffentliche Methoden zum Einsatz.

Entwurf von oben nach unten

Beim Entwurf einer Architektur geht man von oben nach unten vor. Die Gesamtaufgabe wird dabei so lange zerlegt, bis die einzelnen Teile klein genug sind und sich nicht weiter zerlegen lassen. Auf der Ebene von

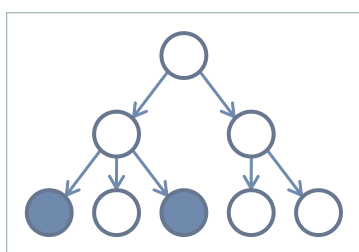


Abb. 1:
Hierarchische Zerlegung von Funktionseinheiten im Entwurf von oben nach unten

Funktionseinheiten startet man in der Regel beim Betriebssystemprozess und zerlegt diesen in Komponenten. Manchmal ist es auch notwendig, oberhalb eines Prozesses zu beginnen, wenn das Problem so groß ist, dass es in mehrere Prozesse zerlegt oder sogar auf mehrere Maschinen verteilt werden muss. Verteilte Anwendungen im Web sind ein Beispiel hierfür.

Ist ein Teilproblem hinreichend komplex, wird es beim Entwurf der Architektur in mehrere Komponenten zerlegt. Oft entsteht auch dabei wieder eine Hierarchie von Komponenten. Eine Komponente kann also gegenüber einer anderen Komponente eine Dienstleistung erbringen, indem sie sich dazu weiterer Komponenten bedient. Dadurch, dass dies hierarchisch erfolgt, sieht die übergeordnete Komponente von diesen Details nichts.

Auf Ebene der Komponenten endet die Aufgabe des Architekten und beginnt die Arbeit des Entwicklers. Seine Aufgabe ist es, die Komponente so in Klassen zu zerlegen, dass die Funktionalität der Komponente hergestellt wird. Und natürlich zerlegt er Klassen in Methoden und diese gegebenenfalls in weitere Methoden.

Single Responsibility Principle

Das Ziel jeglicher Zerlegung ist eine Struktur, die leicht verständlich ist. Entstehen dabei Funktionseinheiten, die keine klar umrissene Aufgabe haben, fällt es schwer diese zu verstehen. Damit wird auch an den Stellen, an denen die Funktionseinheit verwendet wird, das Verständnis erschwert. Verwendet eine Methode beispielsweise eine andere Methode, die mehr als eine Verantwortlichkeit hat, ist es auch schwierig, den Verwender dieser Methode zu verstehen. Ein Verstoß gegen das Prinzip hinterlässt sozusagen bei allen Verwendern auch noch seine Spuren.

Schon die Tatsache, dass man für eine Funktionseinheit nicht schnell einen guten Namen findet, deutet darauf hin, dass die Aufgabe der Funktionseinheit nicht klar abgegrenzt ist. Funktionseinheiten müssen also

eine einzige Verantwortlichkeit haben. Andernfalls leidet das Verständnis. In der Folge ist es dann schwerer, die Korrektheit der Funktionseinheit sicherzustellen, wenn die Einheit für mehrere Dinge verantwortlich ist. Das gleiche gilt für die Evolvierbarkeit. Auch diese leidet, wenn eine Funktionseinheit mehr als eine Verantwortlichkeit hat. Das dieses Problem lösende Prinzip wird als »Single Responsibility Principle« bezeichnet und besagt, dass eine Funktionseinheit nur eine Verantwortlichkeit haben sollte.

Dieses Prinzip lässt sich auch umkehren: Jede Verantwortlichkeit sollte an genau einer Stelle realisiert werden. Wird eine Verantwortlichkeit auf mehr als eine Funktionseinheit verteilt, droht Gefahr. Erfolgt die Verteilung hierarchisch, im Sinne einer Verfeinerung, ist natürlich alles in Ordnung. Wird also die Gesamtfunktionalität in der Hierarchie nach unten weiter untergliedert, entsteht eine verständliche Struktur, weil die Zerlegung hierarchisch erfolgt (siehe Abbildung 1). Es handelt sich dann ja im Sinne der Abstraktion um eine Verfeinerung der Verantwortlichkeit. Wird die Funktionalität jedoch nicht hierarchisch aufgeteilt, ist nicht so leicht erkennbar, wie die Aufteilung erfolgt. In der Folge lässt sich die Funktionalität später nur schwer ändern oder ergänzen, schon weil es schwer ist, alle davon betroffenen Funktionseinheiten zu identifizieren (siehe Abbildung 2).

In der Praxis stellt diese Forderung eine große Herausforderung dar. Dies hängt damit zusammen, dass Ver-

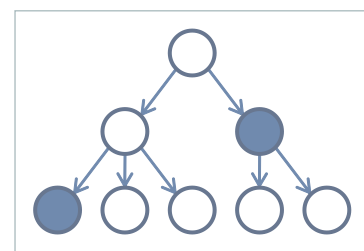


Abb. 2:
Unsaubere hierarchische Zerlegung erschwert Identifikation von Funktionseinheiten

antwortlichkeiten (»Responsibilities«) oft aus mehreren Belangen (»Concerns«) bestehen. So enthält beispielsweise die Verantwortlichkeit »Überweisung von Konto A auf Konto B« mehrere Belange. Einer davon ist die eigentliche Logik einer Überweisung: das eine Konto wird belastet, dem anderen Konto wird der Betrag gutgeschrieben. Neben diesem Belang kommen weitere hinzu: die Überweisung soll so dokumentiert werden, dass später beispielweise nachvollziehbar ist, wer sie veranlasst hat. Dieser als »Auditing« bezeichnete Belang kann nun aber nicht einfach in der Funktionseinheit realisiert werden, die schon für die Überweisung zuständig ist. Denn dann hätte diese Funktionseinheit mehr als eine Verantwortlichkeit. Ferner wird der Belang »Auditing« an mehreren Stellen benötigt, so dass dieser Belang dadurch an mehrere Stellen verteilt würde. Mit den Mitteln der Objektorientierung allein ist diese Herausforderung allerdings nicht zu meistern, sondern es bedarf ergänzend der »Aspektorientierung«.

Single Level of Abstraction

Ein weiteres Prinzip sorgt für leicht verständliche Hierarchien von Funktionseinheiten, nämlich das Prinzip, jeweils die gleiche Ebene der Abstraktion zu verwenden. Am einfachsten ist dies bei der Untergliederung einer Methode zu verstehen. Innerhalb einer Methode soll sich alles auf einem einheitlichen Abstraktionsniveau befinden. Enthält eine Methode Aufrufe anderer Methoden, sollte sie nicht gleichzeitig auch Anweisungen

enthalten, da diese auf einem anderen Abstraktionsniveau liegen. Dies erschwert die Lesbarkeit und damit die Verständlichkeit.

Im Idealfall kann eine Klasse von oben nach unten wie ein Zeitungsartikel gelesen werden. Der Name der Klasse ist sozusagen die Überschrift des Artikels. Es folgen dann die öffentlichen Methoden, die quasi eine

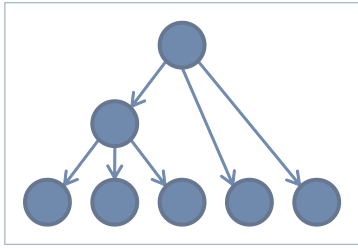


Abb. 3:
Unausgewogenes Abstraktionsniveau des Codes einer Methode

knappe Zusammenfassung des Inhalts geben. Bin ich daran interessiert zu verstehen, wie eine dieser Methoden arbeitet, möchte ich in die Methode reinschauen und dort eine überschaubare Anzahl von Zeilen vorfinden. Dadurch ist es zunächst schon mal möglich, die wenigen Zeilen auf einen Blick zu erfassen.

Neben einer überschaubaren Anzahl von Zeilen ist es für ein leichtes Verständnis hilfreich, wenn sich die Zeilen alle auf dem gleichen Abstraktionsniveau befinden. Stehen dort Methodenaufrufe und Anweisungen vermischt, wird das Verständnis erschwert. Einerseits ein Methodenaufwurf, der möglicherweise weiter untergliedert ist, andererseits eine Anweisung die sich um ein

kleines Details kümmert – das ist unausgewogen (siehe Abbildung 3). Ein einheitliches Abstraktionsniveau führt zu Gleichmäßigkeit in der Hierarchie, die damit leichter verständlich wird (siehe Abbildung 4).

Fazit

Abstraktion durch hierarchische Strukturen ist eines der wichtigsten Konzepte zur Beherrschung von Komplexität. Die Zerlegung in eine hierarchische Struktur muss dabei jedoch zwei elementare Prinzipien einhalten. Zunächst muss jede Funktionseinheit über genau eine Verantwortlichkeit verfügen. Ferner muss die weitere Untergliederung einer Funktionseinheit in Untereinheiten so erfolgen, dass die dabei verwendeten Untereinheiten auf demselben Abstraktionsniveau liegen. Werden beide Prinzipien befolgt, entstehen Strukturen

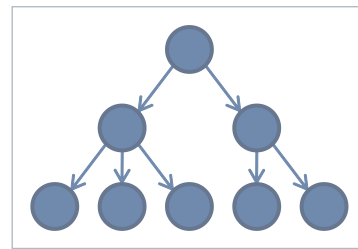


Abb. 4:
Einheitliches Abstraktionsniveau führt zu leichter Verständlichkeit

die leicht verständlich sind. Dadurch können sie in der Zukunft leicht geändert und erweitert werden. Teilen und Herrschen hört sich zunächst einfach an, doch der Teufel steckt, wie so oft, im Detail ... ■

-Anzeige-

ppedv Akademie

Zertifizierung berufsbegleitend

Microsoft
CERTIFIED
IT Professional

Server Administrator

Microsoft
CERTIFIED
IT Professional

Business Intelligence Developer

Werden Sie *Microsoft Certified Server Administrator* oder *Business Intelligence Developer* an den ppedv-Standorten Karlsruhe und Leipzig!

Buchen Sie bis zum 31. Mai 2010 mit Promo-Code **Akademie_VS1**, und Sie erhalten 1 Vollversion Windows Server 2008 R2 oder 1 SQL Server 2008 Standard Edition *gratis* dazu!

MICROSOFT Certified

Weitere Informationen, Termine & Anmeldung unter:
www.ppedv.de/Akademie

**xtopia[kompakt]
Roadshow 2010**

Auch in diesem Frühjahr tourt Microsofts xtopia mit einer Reihe von Informationsveranstaltungen durch Deutschland.



Neben interessanten Vorträgen von Microsoft-Sprechern zu neuen Technologien und Innovationen werden verschiedene namhafte Agenturen das Thema „Web- und Interface Design und User Experience“ aus der Praxis vorstellen. Außerdem werden bekannte Unternehmen über die Verwendung von Interface-Design-Werkzeugen sprechen.

Termine und Locations:

19.04.2010
München
Hilton München Park

20.04.2010
Karlsruhe
Best Western

21.04.2010
Bad Homburg
Maritim

26.04.2010
Köln
Leonardo Hotel

27.04.2010
Hamburg
Baseler Hof

28.04.2010
Berlin
Mövenpick

Die Veranstaltungen beginnen jeweils um 17 Uhr und enden etwa gegen 22 Uhr. In den Pausen und im Anschluss an die Veranstaltung gibt es Gelegenheit, sich untereinander auszutauschen.

Info:

blogs.msdn.com/riablog/archive/2010/01/18/auf-die-pl-tzefertig-anmelden-die-xtopia-kompakt-roadshow-2010.aspx

Welcher Artikel gefällt Ihnen am besten?

Sagen Sie uns Ihre Meinung unter:

www.VisualStudio1.de/Umfrage.aspx