

## Schöner frickeln



**S**oftwareentwicklung braucht Zeit. Welch tolle Erkenntnis. Doch der Kunde will die Software so schnell wie möglich. Auch klar. Von Auftrag bis Auslieferung gibt es eine Menge X an Zeit, die es aufzuteilen gilt. Im Feature-based Programming (siehe dotnetpro.tv auf Seite 60 und auf der Heft-CD) wird eine Min-Max-Schätzung abgegeben, wie lange man für ein Feature voraussichtlich braucht. Die Addition dieser Minima und Maxima für alle Features ergibt dann einen Rahmen für die Gesamtzeit, die für die Anwendungsentwicklung veranschlagt werden muss. Eine gute Sache, da somit die Aufgabe „baue Anwendung mit XX Eigenschaften“ auf die kleinsten für den Kunden sichtbaren Einheiten heruntergebrochen wird.

Was der Ansatz aber nicht beantworten kann, ist, wie viel Zeit Sie für ein Feature oder eine Funktion einplanen sollten. Aktion „Schöner Code“ gegen „Was interessiert mich morgen mein Gefrickel von gestern“. Welcher der Ansätze im Alltag gewinnt, dürfte klar sein. Oder doch nicht?

Wie sieht es mit der Wiederverwendbarkeit aus? Normalerweise sollte diese nicht auf Codeebene passieren, aber auch das ist wieder so ein hehres Ziel, das im Alltag oft schnell über Bord geworfen wird. Also wird doch alter Code ins neue Projekt kopiert. Ach hätte man sich doch nur damals mehr Zeit gelassen und alles sorgfältiger gemacht. Sauber kommentiert und mit einer Doku versehen. Ja, wo leben wir denn? Und doch. Immer wieder wird man mit der Nase darauf gestoßen, dass man doch mehr Zeit in die Wiederverwendbarkeit hätte investieren sollen.

Bei der Softwareentwicklung wird man für das Ergebnis bezahlt. Je schneller also ein Projekt fertig gestellt ist, um so mehr verdient die Firma.

Wer für ein halbes Projekt wegen exzessiver Sorge um die Wiederverwendbarkeit so lange braucht wie ein anderer für das ganze, hat auch halb so viel Verdienst. Zumindest in diesem Projekt. Denn im nächsten Projekt lässt sich der Code vielleicht wieder verwenden. Und dann ist man doppelt so schnell wie der Frickler.

Doch noch eine Frage stellt sich: Auf welche Methoden oder Klassen sollte man besonders viel Sorgfalt und somit Zeit verwenden? Auf den Persistence Layer, weil auf ihm alles ruht? Auf die Logik, weil eine falsche Logik nur Unfug treibt? Auf die Klassen, die für die sichere Authentifizierung sorgen, weil ohne Sicherheit heute nichts mehr Bestand hat?

Es bleibt immer der Widerstreit von langer Überlegung und Recherche gegen schnell erreichte Funktionalität: hervorragende Performance und wenige Zeilen Code gegen mäßige Performance und viele Zeilen Code. Das Dumme ist, dass der Kunde nicht dafür bezahlt, besonders schönen und kurzen Code zu erhalten. Die Anwendung muss das tun, wofür sie gedacht ist, und sollte nicht zu langsam sein. Doch wer Zeit in das Design und einige wichtige Klassen steckt, sorgt für gute Wartbarkeit des Codes und auch für eine gute Performance der gesamten Anwendung. Sie müssen nur noch entscheiden, welche Klassen wichtig sind.

Viel Spaß mit dieser dotnetpro wünscht Ihnen

Tilman Börner  
Chefredakteur dotnetpro