Tolle Planwirtschaft

Früher war alles besser. Jaja. Vor allem aber gab es früher einen Plan. Den Plan von einer Software. Der wurde nach vielen Gesprächen mit dem Kunden akribisch erstellt.

Dann wurde endlos lange implementiert und am Ende kam etwas heraus, das ein bisschen, aber eben nicht ganz etwas anderes tat als das, was der Kunde erwartet hatte. Es gab aber auch Projekte, die es gar nicht über das Pflichtenheft hinaus schafften. Andere wiederum versandeten, weil sie sich als viel zu komplex und damit als nicht mehr umsetzbar erwiesen.

Vielleicht waren es diese Projekte, die zu einem Umdenken bei der Vorgehensweise animierten. Statt des einen großen Fünfjahresplans sollten die Anwendungen Stück für Stück abgeliefert werden. Verbunden mit dem Einbeziehen des Kunden alle paar Wochen war so die agile Vorgehensweise geboren. Diese Art der Entwicklung hatte zwei gravierende Vorteile: Die kürzeren Zeitabschnitte sind für den Menschen besser einschätzbar und zwingen zur Reflexion: Was ist schon geschafft? Wo hängt es?

Und sie versetzen den Kunden in die Lage, sofort zu meckern und darauf zu bestehen, dass die Datenbank auf jeden Fall rosa sein muss.

Trotz dieser Sprints genannten kurzen Abschnitte sollte in groben Zügen klar sein, was für eine Anwendung das wird und wie deren Architektur aussieht. Es sollte also Konsens über die Grundzüge der Anwendung bestehen. Der allumfassende Plan aber ist von der Dauer her kürzer geworden und erstreckt sich nur noch von einem Sprint zum nächsten.

Führt man die Entwicklung logisch weiter und verkürzt den Plan noch weiter, kommt man in den Bereich der Minutensprinter. Grober Plan steht („Irgendso ein Datenverwaltungstool"), und schon geht es los mit den Sprints: Implementiere Zeile, siehe, sie war gut. Wenn nicht, editiere Zeile, beginne von vorn.

Böse Zungen behaupten, dass das die Trial-and-Error-Methode sei, aber es gibt ja immer Übelkrähen, die alles schlechtreden. Der Vorteil der Methode liegt auf der Hand: Statt aufwendig Tests zu schreiben, erfährt der Entwickler unmittelbar, was er für Fehler in den Code gemurkst hat. Freilich dauert diese Vorgehensweise genauso lange, wie Tests zu schreiben. Dafür funktioniert sie aber wenigstens weder automatisch, noch ist sie nachhaltig.

Ich wünsche Ihnen wertvolle Informationen aus der dotnetpro!

Tilman Börner
Chefredakteur dotnetpro

David Tielke

stellt klar, warum .NET Core kommen musste (S. 20)

Marius Schulz

klärt auf, was sich alles bei TypeScript 2.0 getan hat (S. 58)

Christine Biswanger

zeigt, wie .NET–Software biometrische Merkmale berücksichtigen kann (S. 70)