

COMPUTERWOCHE

■ DG B 2615 C 31. JAHRGANG

NACHRICHTEN • ANALYSEN • TRENDS

Abstraktion hilft Softwarekomplexität zu beherrschen

Grenzen der grafischen Modellierung

„Das Pferd ist tot! Nun, dann wechseln wir den Reiter“ – Witze über Softwareprojekte gibt es zuhauf. Und sie sind beliebt, helfen sie doch, den alltäglichen Irrsinn bei der Entwicklung von Software zu ertragen. Doch lustig ist die Situation keinesfalls. Einen Ausweg aus dieser Misere bietet die grafische Modellierung mit der Unified Modelling Language (UML) – wenn gewisse Regeln beachtet werden.

Von Martina Maier*

VIELE IT-PROFIS kennen aus eigener Erfahrung Projekte, die nach dem Motto „Nur das Genie beherrscht das Chaos“ organisiert wurden oder schlicht die Business-Anforderungen des Kunden

den nicht erfüllt haben. Projekte, die Zeitplan und Budget weit überzogen oder bei denen die geplanten Anwendungen nie das Licht der Welt erblickten.

Warum lernt man so wenig aus den Fehlern der Vergangenheit? Ein wesentlicher Grund liegt sicher im Produkt selbst. Software besteht in letzter Konsequenz aus einer riesigen Zahl von Codezeilen. Diese Form der Anwendungsbeschreibung kann für kleinere Projekte ausreichend sein. Schwieriger wird es, wenn der Programmcode sehr umfassend wird. Denn hier gibt es keine abstrakteren Zwischenmodelle, die eine schrittweise Näherung an die Anwendung ermöglichen. Unterschiedliche Sichten des Systems aus verschiedenen Blickwinkeln und Abstraktionsebenen werden nicht unterstützt.

Noch schwieriger wird es, wenn der Code über längere Zeit von unterschiedlichen Entwicklern ge-

pfligt wurde. Wartung und Erweiterungen beeinträchtigen die strukturelle Qualität der Software zunehmend. Es ist dann nahezu unmöglich, sich aus zehntausenden Zeilen Code einen Überblick über die Struktur der Anwendung und die in ihr abgebildeten Beziehungen zu verschaffen. Textliche Beschreibungen wie Programmcode sind also eine denkbar ungeeignete Arbeitsgrundlage für den effizienten Umgang mit einer Softwareanwendung.

Dolmetschen ist fehlerträchtig

Der fachlich Verantwortliche einer Anwendung beherrscht nur selten Sprachen wie Java oder C++. Er hat damit keine reelle Chance, sich in das Anwendungssystem selbst „einzulesen“ oder sich einen Überblick über den aktuellen Stand zu verschaffen. Er benötigt den Entwickler als Dolmetscher und ist abhängig von dessen Kenntnissen und Fähigkeiten. Doch Dolmetschen ist fehlerträchtig, und oft kommt in der Übersetzung etwas Sinnverfälschtes heraus. Die zentrale Herausforderung in der Softwareentwicklung ist deshalb der Umgang und die Beherrschung der damit verbundenen Komplexität.

Ein Blick über den Tellerrand zeigt, wie es gehen kann: Die Entwicklung von Maschinen etwa wurde durch die Einführung von CAD-Systemen, die eine zwei- und dreidimensionale Konstruktion ermöglichen, deutlich vereinfacht. Das Ergebnis sind kürzere

Hier lesen Sie ...

- warum so viele klassische Softwareprojekte scheitern;
- welche Probleme der Wechsel von der textlichen Programmierung zur grafischen Modellierung beheben soll;
- wo bisher die Schwächen der grafischen beziehungsweise UML-gestützten Modellierung liegen;
- worauf das Projekt-Management achten sollte, wenn es die mit den UML-Defiziten verbundenen Stolpersteine meiden will.

Entwicklungszeiten, höherwertige Lösungen und geringere Entwicklungskosten. Der entscheidende Entwicklungsschub lag in der deutlich besseren Beherrschbarkeit von Komplexität. Auch in der Softwareentwicklung führt dauerhaft kein Weg an der grafischen Modellierung vorbei. Ein erster Schritt in diese Richtung ist mit der Modellierungssprache Unified Modelling Language (UML) getan - grafisches Modellieren wird im Folgenden gleichgesetzt mit UML-gestütztem Vorgehen und umfasst auch den Einsatz entsprechender Modellierungs-Tools.

UML ist eine Sprache, deren Wortschatz aus grafischen Elementen besteht. Wer UML „spricht“, entnimmt aus UML-Grafiken sehr schnell die Struktur und Funktion von Anwendungsspekten. Die grafische Darstellung von Modellen verschafft einen schnellen Überblick über einzelne Komponenten der Anwendung und deren Strukturen. So erlaubt beispielsweise die Modellierung von Klassendiagrammen eine einfach verständliche Darstellung von Business-Klassen und deren wechselseitigen Abhängigkeiten. UML bietet zudem eine abstraktere Darstellung als Programmcode, da Klassen nur mit ihren Methodensignaturen dargestellt werden. Die für eine erste Systemübersicht irrelevante Methodenimplementierung wird nicht dargestellt. Auch die Sprachbarrieren zwischen Fachabteilung und Entwicklern lassen sich überbrücken. Für Fachleute ist das Erlernen einer grafischen Modellierungssprache deutlich einfacher als das einer herkömmlichen Programmiersprache. Die leicht einprägsame Symbolik unterstützt das Verständnis von einzelnen Modellen.

Transformation bis zur implementierten Lösung

Für den effizienten Einsatz von grafischer Modellierung ist es sinnvoll, dass sie breitflächig in allen Phasen der Entwicklung eingesetzt wird. Dies fängt bei der Analyse an und geht über das Design bis hin zur Implementierung. Die phasenübergreifende Modellierung ist beispielsweise die Grundlage der Model Driven Architecture (MDA). Ein zentraler Aspekt der MDA ist die Transformation von Modellen. Ausgehend

UML: Pro und Kontra

- ⊕ Von Fachexperten leicht zu erlernen;
- ⊕ reduziert Komplexität;
- ⊕ ermöglicht eine schrittweise Annäherung an und unterschiedliche Sichtweisen auf die zu entwickelnde Anwendung;
- ⊕ mit Hilfe eines geeigneten Vorgehensmodells sind Kosteneinsparungen möglich;
- ⊕ Softwareentwicklung bleibt transparent;
- ⊕ bessere Dokumentation der Entwicklung;
- ⊕ Mehrwertdienste können einfach integriert werden.
- ⊖ Medienbrüche;
- ⊖ keine Modellierung von Methoden;
- ⊖ noch kein ausreichender Sprachschatz;
- ⊖ beispielsweise keine Beschreibung von Benutzeroberflächen.

von einem plattformunabhängigen Anfangsmodell, werden durch schrittweise Umformungen plattformspezifischere Modelle erzeugt. Das Endmodell spiegelt die implementierte Lösung wider.

Für die konkreten Modellierungsformen in den einzelnen Phasen – also welcher Aspekt wird mit welchen Sprachmitteln formuliert - gibt es bislang keine Standards. In der Analyse können die fachlichen Anforderungen unter anderem in Form von Use-Cases, Klassendiagrammen und Activity-Diagrammen modelliert werden. Leider gibt es keine geeignete Ausdrucksform für die Beschreibung von Benutzeroberflächen. Und dies, obwohl gerade die Oberflächen dringend notwendig sind, um dem fachlichen Anwender die spezifizierte Lösung näher zu bringen.

Im Design hat die Verwendung von grafischer Modellierung unterschiedliche Spannweiten. Es gibt Ansätze zur vollständigen Modellierung der Model-View-Controller-Schichten und Ansätze, in denen ausschließlich auf der Ebene der Business-Logik Modellierung stattfindet. Die Entscheidung, die View- und Controller-Schicht zu modellieren, ist oft von zwei Fragen abhängig:

- Wird ein Produkt oder eine Individuallösung entwickelt?
- Gibt es für die Lösung verschiedene GUI-Zielpattformen?

Ein UML-Modell ist nur hilfreich, wenn es den aktuellen Implementierungsstand widerspiegelt. Dies war in den Anfängen der grafischen Modellierung oft nicht der Fall. Es wurde ein UML-Modell entworfen und losgelöst davon dessen Umsetzung programmiert. Damit war das Auseinanderdriften von grafischem Modell und Code vorbestimmt. Der Nutzen des grafischen Modells ging verloren, es wurde zur Seite gelegt und nicht mehr gepflegt. Die Ursache lag darin, dass die Tools zur Modellierung in UML noch nicht den entsprechenden Reifegrad hatten und keine Exportmöglichkeiten oder bedarfsgerechten APIs zur Verfügung stellten. Damit war der Weg zur Verbindung von Modell und Programmcode versperrt. Das hat sich inzwischen geändert: Verschiedene Hersteller bieten Modellierungs-Tools an, die sich für unterschiedliche Anforderungen eignen.

Der Weg zur grafischen Programmiersprache

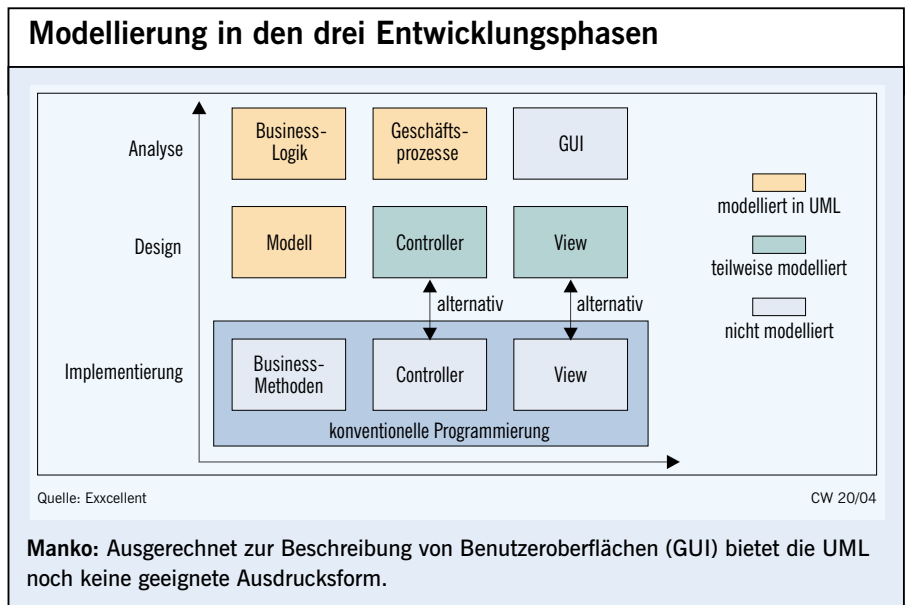
Es wäre jedoch zu einfach, den mangelnden Reifegrad der grafischen Modellierung allein unzulänglichen Tools anzulasten. Um die Effizienz zu steigern, muss sich UML als grafische Programmiersprache qualifizieren. Damit ist Folgendes gemeint: Heute geschieht in ähnlicher Weise, was sich vor rund 20 Jahren mit der Abkehr von der maschinennahen Programmiersprache Assembler zu C und später zu objektorientierten Sprachen vollzogen hat. Wir befinden uns mitten im Umbruch von der textlichen zur grafischen Programmierung. Bei diesem Wechsel kommt allerdings ein wichtiger Aspekt hinzu. Die neue „Programmiersprache“ UML hat keinen ausreichenden Sprachschatz, um insbesondere Implementierungsdetails adäquat ausdrücken zu können. Auch die Erweiterungen von UML 2.0 bringen keine zufrieden stellende Lösung. Solange es aber keine Sprache gibt, in der eine vollständige Ausdrucksmöglichkeit von Softwarelösungen besteht, muss man in mindestens zwei Sprachen sprechen.

Dieser Systembruch führt zu größerer Komplexität bei der Projektorganisation. Mängel in dieser Organisation können die Vorteile der grafischen Modellierung überwiegen oder zunichte machen. Eine vollständige und ausdrucksstarke grafische Beschreibungssprache würde uns von dem Sprachbruch befreien.

Mit Codegenerierung kombiniert

Die daraus resultierenden Perspektiven sind viel versprechend. Zum Beispiel bestünde die Möglichkeit, das grafische Modell um fachliche und technische Eigenschaften zu erweitern, die von Generatoren interpretiert und umgesetzt werden. Modellierung würde also mit Codegenerierung kombiniert.

Erweiterungen können so weit gehen, dass man von einem eigenen „Dialekt“ sprechen kann. Nehmen wir zum Beispiel die Änderungsverfolgung einer Person in Bezug auf ihre Bankverbindung. Zu jeder Person sollen Änderungszeitpunkte von Kontoangaben und vorliegenden oder nicht mehr bestehenden Lastschriftermächtigungen nachvollzogen werden können. Die Implementierung dieses Features ist sehr aufwändig. Die Nachvollziehbarkeit der Änderungen muss gesichert sein, ebenso ist das Schreiben der Historie an verschiedenen Stellen zu berücksichtigen. Im Fall des Modellierungs- und Generierungsansatzes genügt es hingegen, die Klasse mit „Änderungsverfolgung“ zu kennzeichnen. Den Rest erledigt der Generator automatisch. Mehrwertdienste können so auf einfache Weise zu- und abgeschaltet werden.



Einige zentrale Fragen bei grafischen Modellierungsansätzen stellen sich für die Projektleitung. Hierzu gehören: Was soll modelliert werden, wie funktioniert Modellierung im Team, und welche Dialekte finden Anwendung? Ein standardisiertes Vorgehen gibt es derzeit nicht. So hängt der erzielte wirtschaftliche Vorteil entscheidend von drei Faktoren ab: erstens von dem zugrunde liegenden Vorgehensmodell und der damit verbundenen Projektorganisation, zweitens von der Nutzung der Perspektiven zur Integration von Mehrwertdiensten und schließlich von der richtigen Tool-Auswahl.

Herausforderung der Projektorganisation

Die größten Stolpersteine beim Umgang mit grafischer Modellierung sind Systembrüche in der Beschreibung der Modelle. Dazu gehört in der Analyse die Spezifi-

kation der Benutzeroberflächen mit ihrer Abbildung auf den Business-Layer. Werden Oberflächen in einer völlig anderen Umgebung beschrieben, ist damit zum einen deren Pflegeaufwand sehr hoch. Zum anderen ist keine Wiederverwendung oder automatisierte Transformation für das Design möglich. Im Design ist aber die effiziente und nahtlose Integration von Modellierung und Codierung entscheidend. Die Erfahrung lehrt, dass Themen wie Paketbildung, Generierungseinheiten und damit Generierungszeiten und nicht zuletzt die Versionskontrolle der Modelle von der Projektorganisation sorgsam bedacht werden sollten. Dann kann grafische Modellierung schon heute wirtschaftliche Vorteile bringen. (ue) ←

*Dr. rer. nat. Martina Maier ist Geschäftsführerin der Exccellent Solutions GmbH in Ulm.

ex|Xcellent
solutions

eXXcellent solutions gmbh
In der Wanne 55
D - 89075 Ulm

t | +49 [0]731-55026 - 0
e | m.maier@excellent.de
i | www.excellent.de