Entwurf und Konstruktion **der Features**

Das erste und wichtigste Element beim Feature Driven Development ist das Feature. Dieser Artikel beschäftigt sich mit Entwurf und Programmierung der Features. Er zeigt das Mapping der Features auf Klassen und Methoden und stellt Code-Inspektionen als Technik zur Qualitätssicherung in FDD vor.

von Stefan Roock und Henning Wolf

Die drei Feature-Driven-Development-Teilprozesse vor der eigentlichen Entwicklung dienen der Abklärung, welche Features umgesetzt werden sollen, und der Formulierung der Features. Diese Teilprozesse sind „Entwickle Gesamtmodell", „Erstelle Feature-Liste" und „Plane je Feature". Dabei geht es jeweils um die Gesamtmenge aller Features des Projekts (Abb. 1). Die hier beschriebenen Teilprozesse „Entwirf je Feature" und „Konstruiere je Feature" werden hingegen immer für ein einzelnes Feature durchlaufen. In der Regel werden sie außerdem hochgradig parallel ausgeführt, sodass immer mehrere Features gleichzeitig in der Entwicklung sind.

Prozess #4: Entwirf je Feature

Im ersten FDD-Teilprozess „Entwickle Gesamtmodell" [1], [2] ist bereits ein Domänenmodell entstanden. Insofern könnte man sich fragen, was denn jetzt noch je Feature an Entwurf fehlt. Dazu sollte man sich in Erinnerung rufen, dass das Domänenmodell eben genau die Funktion hat, einen Modellierungsüberblick zu geben. Dort werden alle Fachkonzepte genannt und in Beziehung zueinander gesetzt. Es wird aber keinesfalls für alle Klassen im Detail geklärt, welche Methoden mit welcher Signatur (Parameter, Typisierung etc.) benötigt werden. Diese Detaillierungsebene leistet der Teilprozess „Entwirf je Feature", in dem die folgenden Aufgaben ausgeführt werden:

1. Stelle Feature-Team zusammen
2. Optional: Gib Domänenüberblick
3. Optional: Studiere Dokumente
4. Entwirf Sequenzdiagramme
5. Verfeinere Klassendiagramme
6. Schreibe Klassen- und Methodenrümpfe
7. Verifiziere Entwurf

Feature-Team zusammenstellen

Das entwickelte Gesamtmodell ist Grundlage des Entwurfs der einzelnen Features, kann dabei aber durchaus noch Veränderungen erfahren. Zusätzlich benötigen wir die Class-Owner-Liste aus dem Teilprozess „Plane je Feature" und die dort ebenfalls vorgenommene Zuordnung der Klassen zu Features. Hieraus ergeben sich die Gruppe der am Entwurf beteiligten Entwickler sowie der zum Feature gehörige Chef-Entwickler. Sie bilden gemeinsam das so genannte Feature-Team, welches sich auf diese Weise dynamisch für jedes Feature bildet. Da parallel mehrere Features gleichzeitig abgearbeitet werden (Abb. 2), kommt es auch oft vor, dass Entwickler gleichzeitig am Entwurf (und der Entwicklung) mehrerer Features beteiligt sind.

Domänenüberblick geben und Dokumente studieren

Je nach Komplexität des Features finden die folgenden beiden Aktivitäten statt:

- Ein Fachexperte gibt einen Überblick über die Domäne mit Bezug auf das aktuell zu entwerfende Feature für das gesamte Feature-Team.
- Das Feature-Team sichtet und liest die referenzierte Dokumentation zum Feature. Das können Fachkonzeptdokumente sein, aber auch Spezifikationen von anzusprechenden Drittsystemen etc.

Sequenzdiagramme entwerfen

Das Feature-Team entwickelt gemeinsam die notwendigen Sequenzdiagramme, um das Feature im System umzusetzen. In diesem Zug wird auch festgelegt,

Navigator FDD-Artikelserie

welche Klassen welche Methoden mit welchen Parametern und Rückgabewerten haben. Das Sequenzdiagramm wird später ins Versionskontrollsystem eingecheckt, dabei werden gegebenenfalls verworfene Designalternativen und bewusst getroffenen Designentscheidungen dokumentiert.

Es mag übertrieben scheinen, für alle Features Sequenzdiagramme zu entwerfen, aber sie haben bei FDD die wichtige Funktion, dass das Feature-Team und damit die einzelnen Class-Owner sehr genau wissen, welchen Teil der Funktionalität sie mit ihren Klassen erbringen müssen und wie sie dabei mit anderen Klassen (und damit auch den Class-Ownern) kooperieren.

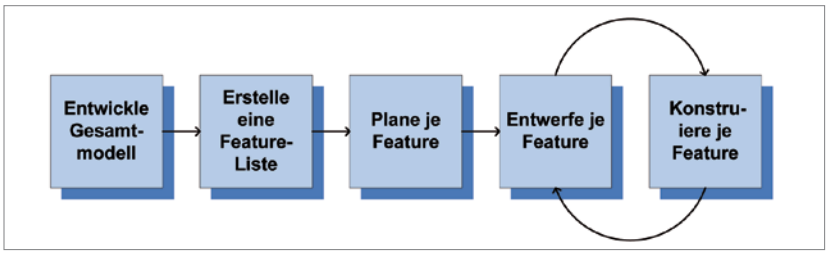

Abb. 1: Die fünf FDD-Prozesse

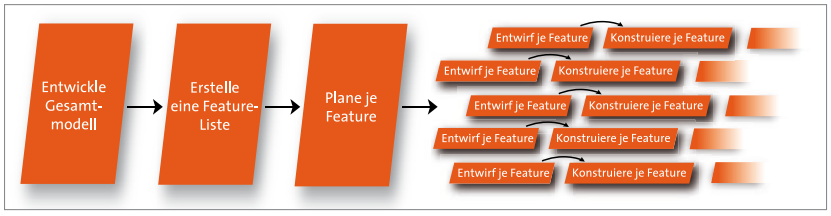

Abb. 2: Parallele Abarbeitung von Features

Klassendiagramme verfeinern

Nachdem die Sequenzdiagramme erstellt und die Methoden für die Klassen festgelegt wurden, verfeinert der Chefentwickler die Klassendiagramme aus den ersten Prozessen und erstellt elektronische Aufbereitungen der Sequenzdiagramme, die im Team meist nur auf Flipcharts erstellt wurden. Je nach verwendetem UML-Tool generiert er gegebenenfalls Klassen- und Methodenrümpfe, auf deren Basis die Entwickler weiterarbeiten. Außerdem erstellt er für das Feature-Team einen speziellen Bereich im Versionskontrollsystem, auf dem diese gemeinsam arbeiten können.

Klassen- und Methodenrümpfe schreiben

Jeder Entwickler schreibt auf Basis der gemeinsamen Vereinbarungen (Sequenzdiagramme!) für seine Klassen und Methoden Rümpfe. Dies betrifft in der Regel die Methodensignaturen, also die Namen der Methoden, die Rückgabeparameter, die Parameterliste mit Typisierung, gegebenenfalls auch Exceptions. Wenn alle Entwickler damit fertig sind, erstellt der Chefentwickler auf dieser Grundlage eine API-Dokumentation, die dem gesamten Entwicklungsteam (also nicht nur dem Feature-Team) zur Verfügung gestellt wird, z.B. in einem Projekt-Intranet.

Entwurf verifizieren

Der Entwurf wird per Design-Inspektion überprüft. Dabei entscheidet der Chefentwickler, ob ihm eine gemeinsame Inspektion im Feature-Team ausreicht oder ob er weitere Projektmitglieder hinzuziehen möchte (z.B. den Chefarchitekten). Eine Design-Inspektion ist ein gemeinsames Treffen, bei dem die Sequenzdiagramme und die entstandenen Klassen- und Methodenrümpfe gemeinsam begutachtet werden. Dabei wird sowohl erneut auf der Ebene des Gesamtzusammenhangs geprüft, ob die Lösung angemessen ist, als auch auf Ebene der Details (Klassen- und Methodenrümpfe) entschieden, ob die Umsetzung gut gewählt ist. Während der Inspektion werden je Klasse gewünschte Veränderungen am Entwurf notiert und der entsprechende Entwickler (Class-Owner) vermerkt sich diese auch auf seiner Aufgabenliste. Wenn der Entwurf verifiziert wurde, kann mit der Implementierung begonnen werden.

Prozess #5: Konstruiere je Feature

Im fünften und letzten Prozess wird das Feature programmiert. Dazu werden die folgenden Aufgaben erledigt:

Objekt-Feature-Mapping beim Entwurf

Die Feature-Form *<Aktion> <Ergebnis> <Objekt>* lässt sich bei objektorientierter Programmierung direkt ins Klassenmodell übertragen: Die Aktion ist die Methode, die am Objekt aufgerufen wird und das Ergebnis liefert. Aus „Berechne Summe der Aufträge“ wird

```
public class Auftraege {
public Betrag berechneSumme() {…}
…
}
```

Diese direkte Verbindung zwischen fachlichen Anforderungen und fachlichem Klassenmodell nennt sich *Strukturähnlichkeit* und hat eine Reihe von positiven Effekten: Wenn sich eine Anforderung ändert, findet man schnell die zu ändernde Stelle. Wenn man eine Anforderung programmiert, kann man leichter abschätzen, ob und wie sich dadurch andere Anforderungen ändern. Und nicht zuletzt wird der Programmcode leichter verständlich, weil er damit in der „Sprache der Anwender“ geschrieben ist.

Das Feature-Team muss aber zusätzlich für jedes Feature klären, wie es letztlich intern implementiert wird. Denn auch wenn die Methode schnell gefunden ist, mit der auf oberster Ebene das Feature erbracht wird, so können doch dahinter beliebig komplizierte Abfolgen von Methodenaufrufen stehen.

1. Implementiere Klassen und Methoden
2. Führe Code-Inspektionen durch
3. Schreibe Unit-Tests
4. Gib Klassen für Build-Prozess frei

Klassen und Methoden implementieren

Auf Basis des gemeinsamen Designs, der Sequenzdiagramme und der Klassen- und Methodenrümpfe implementiert jeder Entwickler die benötigte Funktionalität für dieses Feature in seinen Klassen.

Code-Inspektionen durchführen

Der Chefentwickler entscheidet, ob Code-Inspektionen vor oder nach dem Erstellen von Unit-Tests durchgeführt werden. Er entscheidet auch, ob die Code-Inspektion nur innerhalb des Feature-Teams erfolgt oder ob weitere Projektmitglieder hinzugezogen werden sollen. Es finden aber in jedem Fall Code-Inspektionen statt, bei denen gemeinsam der Code inspiziert und festgehalten wird, was am Code noch verbessert werden muss. Dies kann sich auf die Länge von Methoden, die Verwendung von Algorithmen, Variablennamen etc. beziehen.

Unit-Tests schreiben

Außerdem werden Unit-Tests für alle Klassen erstellt. Sie dienen insbesondere der Vermeidung unerwünschter Nebeneffekte bei späteren Änderungen und Erweiterungen der Klassen. Ob für ein Feature auch über einzelne Klassen hinaus integrierende Unit-Tests erstellt werden, entscheidet der Chefentwickler.

Klassen für Build-Prozess freigeben

Wenn die Entwickler ihre Klassen nach der Inspektion und der Erstellung von Unit-Tests fertig melden, entscheidet der Chefentwickler, welche Klassen er für den Build-Prozess freigibt. Er bildet damit die Klammer über die gesamte Funktionalität des Features. Er leistet bei Bedarf die noch notwendige Integration und checkt den Code in das Projekt-Versionssystem ein.

Bewertung

Die Entwürfe so explizit vor der Implementierung zu erstellen, kommt schon sehr klassisch daher und wirkt wenig agil. Allerdings sollte man dabei bedenken, dass zum einen das gesamte Team gemeinsam entwirft und sich damit alle einbringen können, zum anderen wird ja keinesfalls der Entwurf des Gesamtsystems vorgenommen, sondern lediglich der Entwurf der relevanten Klassen und Methoden für jeweils ein Feature. Damit unterscheidet es sich deutlich von klassischen Ansätzen und rückt rüber in die agile Ecke. Dieses FDD-Vorgehen ist insbesondere dann gut geeignet, wenn in einem Projekt wenige erfahrene (Chef-)Entwickler mit vielen weniger erfahrenen Entwicklern zusammenarbeiten. Der explizite gemeinsame Entwurf der Sequenzdiagramme reduziert die Gefahr, dass sich einzelne Entwickler in Sackgassen manövrieren. Außerdem behält der Chefentwickler immer den Überblick über den Gesamtentwurf und kann früh erkennen, wenn sich Integrationsprobleme zwischen Einzelteilen abzeichnen. Die agilen Methoden neigen mitunter dazu, das vorhandene Erfahrungs- und Qualifikationsgefälle

Diskussion Inspektionen vs. Pair-Programming vs. TDD

Bei den Mechanismen zur Qualitätssicherung gibt es große Unterschiede zwischen den verschiedenen agilen Methoden. Während FDD auf Code-Inspektionen und nachträglich geschriebene Unit-Tests setzt, verwendet XP testgetriebene Entwicklung mit Test-First und Pair-Programming.

Bei der *testgetriebenen Entwicklung* beginnt man stets mit dem minimalen Test, der zunächst fehlschlägt. Anschließend programmiert man genau so viel Produktionscode, bis der Test erfolgreich durchläuft. Den erfolgreichen Test verwendet man als Sicherheitsnetz, um den gerade produzierten Code in eine optimale Form zu bringen (Refactoring). Anschließend schreibt man den nächsten minimalen Test usw. Über diesen Ansatz entsteht der Entwurf des Systems inkrementell über die vielen kleinen Refactoringschritte.

Beim *Pair-Programming* arbeiten stets zwei Entwickler zusammen an einem Rechner. Dabei diskutieren sie die ganze Zeit über den Entwurf und das, was sie programmieren. Sie tauschen alle paar Minuten die Tastatur, sodass beide Partner die ganze Zeit über aktiv an der Programmierung teilnehmen.

Trotz der Unterschiede im Vorgehen laufen die Ansätze von FDD und XP auf Ähnliches hinaus. Pair-Programming kann man als eine spezielle Form der Code-Inspektion verstehen. Es nehmen jeweils nur zwei Entwickler teil, es findet die ganze Zeit über statt und es ist mit der eigentlichen Programmierung verwoben. Testgetriebene Entwicklung erzeugt genauso Unit-Tests wie das nachträgliche Schreiben. Für die Absicherung gegen unbeabsichtigte Nebeneffekte laufen beide Testansätze also auf das Gleiche hinaus – man kann allerdings argumentieren, dass testgetriebene Entwicklung zu besser testbarem Code und einer besseren Testabdeckung führen kann. Ähnlich wie beim Pair-Programming vermischt die testgetriebene Entwicklung das Erstellen der Tests mit dem Entwurf und der Implementation der produktiven Klassen.

Insgesamt setzen Pair-Programming und testgetriebene Entwicklung also darauf, die qualitätssichernden Aktivitäten direkt mit Entwurf und Implementierung zu verzahnen. Das kann effektiver sein, weil die Feedbackschleifen kürzer sind. Diesen Vorteil bezahlt man aber mit höherem Lernaufwand und der Gefahr, dass der Prozess „verschludert“ und letztlich nur noch programmiert, aber nicht mehr qualitätsgesichert wird.

Qualitätssicherung immer notwendig

Qualitätssicherung – in welcher Form auch immer – ist auch und vor allem in agilen Projekten wichtig.
Jedes erfolgreiche Projekt „lebt“ und muss nach der Inbetriebnahme immer wieder geändert und erweitert werden. Agil entwickelte Software zeichnet sich dadurch aus, dass sie von vornherein stärker auf dieses „Leben nach dem Projekt“ vorbereitet wird. Beginnen wir mit Entwurf und Programmierung eines Features, müssen wir häufig existierenden Code anpassen. Das funktioniert nur dann mit vertretbarem Risiko und geringen Kosten, wenn der existierende Code entsprechend vorbereitet wurde: er ist einfach, klar, verständlich, redundanzfrei und mit umfangreichen automatisierten Tests versehen.

zwischen Entwicklern zu ignorieren. XP erweckt beispielsweise leicht den Eindruck, man könne einfach alle Entwickler als gleich betrachten. Das gilt aber eben nur so weit, wie alle Entwickler auch die XP-Techniken beherrschen, insbesondere die testgetriebene Entwicklung (und das ist alles andere als trivial).

Übertragung

Die hier vorgestellte FDD-Vorgehensweise bei Entwurf und Implementierung lässt sich z.B. nur für ausgewählte Features in Nicht-FDD-Projekten einsetzen. Wo allerdings keine individuelle Class-Ownership existiert, müsste diese zumindest für ein Feature hergestellt werden, was eher unnatürlich wirkt. Da scheint es schon realistischer, dass man gegebenenfalls Teams oder Teilteams bildet, in denen so wie hier beschrieben vorgegangen wird, inklusive individueller Class-Ownership. Das ist insbesondere dann eine gute Idee, wenn Entwickler mit weniger Entwurfserfahrung eingebunden werden.

Was die Qualitätssicherung angeht, so können Code-Inspektionen und Pair-Programming sich wechselseitig ersetzen oder ergänzen. Ein Projekt, das beispielsweise XP praktiziert, jedoch Probleme mit dem Pair-Programming hat, dürfte von den Code-Inspektionen à la FDD profitieren. Allerdings müssen die Code-Inspektionen immer konsequent direkt nach der Implementierung jedes Features durchgeführt werden. Ansonsten besteht die große Gefahr, dass erst die festgestellten Nacharbeiten auf die lange Bank geschoben und schließlich die Code-Inspektionen komplett ausgelassen werden.

Wir haben FDD in einem Projekt erfolgreich mit testgetriebener Entwicklung kombiniert. Das funktioniert allerdings nur, wenn Class-Ownership weniger streng gehandhabt wird. Schließlich nimmt man beim testgetriebenen Entwickeln typischerweise auch am „Rande“ seiner Zuständigkeiten Refactorings vor. Das gemeinsame Erstellen der Sequenzdiagramme und die Vorab-Modellierung der Klassen im Detail verschwinden ganz. Genau dieser Aspekt des Entwurfs wird ja gerade durch die testgetriebene Entwicklung abgedeckt.

Ausblick

Der vierte und letzte Teil der Artikelserie zeigt, wie das Projektmanagement im Feature Driven Development abläuft und welche Steuerungs- und Controllinginstrumente verwendet werden.


Stefan Roock und **Henning Wolf** sind Berater bei der akquinet it-agile GmbH. Sie arbeiten als Entwickler, Projektleiter und agile Methodenberater. Sie haben seit 1998 Erfahrungen mit agilen Methoden. Über XP haben sie ein Erfahrungsbuch geschrieben, in Scrum und FDD sind sie zertifiziert.

Links & Literatur

[1] Stefan Roock, Henning Wolf: Feature-Driven-Development, Teil 1: Überblick, in: *Java Magazin* 01.2008, S. 118–121

[2] Thilo Fotscher: Mit Color Modeling farbige Domänenmodelle erstellen, in: *Java Magazin* 12.2007, S. 37–42

[3] Stefan Roock, Henning Wolf: Feature-Driven-Development, Teil 2: Planung mit Featurelisten, in: *Java Magazin* 02.2008, 126–129

[4] Henning Wolf, Stefan Roock, Martin Lippert: eXtreme Programming, 2. Aufl., dpunkt, 2005

[5] FDD-Beschreibung auf 10 Seiten: www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf

[6] Stephen R. Palmer, John M. Felsing: A Practical Guide to Feature Driven Development, Prentice Hall International, 2002

Impressum

Verlag:
Software & Support Verlag GmbH

Anschrift der Redaktion:
Java Magazin
Software & Support Verlag GmbH
Geleitsstraße 14
D-60599 Frankfurt am Main
Tel. +49 (0) 69 6300890
Fax. +49 (0) 69 63008989
redaktion@javamagazin.de
www.javamagazin.de

Chefredakteur: Sebastian Meyen
Redaktion: Claudia Schaumlöffel
Chefin vom Dienst: Nicole Bechtel
Schlussredaktion: Nicole Bechtel, Katharina Klassen, Frauke Pesch
Leitung Grafik & Produktion: Jens Mainz
Layout, Titel: Kristin Brockmann, Jessica Demirkaya, Melanie Hahn, David Heinze, Dominique Kalbassi, Jens Mainz, Michel Michiels-Corsten, Katharina Ochsenhirt, Maria Rudi
CD/DVD-Erstellung: Daniel Zuzek

Autoren dieser Ausgabe:
Raoul Becke, Axel Böttcher, Matthias Bohlen, Andy Bosch, Ben Chelf, Markus Demolsky, Agim Emruli, Jörg Endres, Alexander Felber, Dr. Falk Fraikin, Benjamin Gehr, John Kodumal, Jürgen Kohl, Dirk Koller, Stefan Krecher, Jan Leßner, Marcel Malitz, Matthias Ostermaier, Roland Petrasch, Thomas Ries, Stefan Roock, Oliver Rummeyer, Bruce Sams, Ralph Soika, Matthias Weßendorf, Harald Wolf, Henning Wolf, Eberhard Wolff

Anzeigenverkauf:
Software & Support Verlag GmbH
Patrik Baumann
Tel. +49 (0) 69 6300890
Fax. +49 (0) 69 63008989
pbaumann@javamagazin.de

Es gilt die Anzeigenpreisliste Nummer 11

Pressevertrieb:
DPV Network
Tel.+49 (0) 40 378456261,
www.dpv-network.de

Druck: PVA Landau
ISSN: 1619-795X

Abo-Service:
Software & Support Verlag GmbH
Tel. +49 (0) 69 6300890
Fax +49 (0) 69 63008989
www.javamagazin.de/service/

Abonnementpreise der Zeitschrift:

Inland:	12 Ausgaben	€ 79,–
Europ. Ausland:	12 Ausgaben	€ 89,–
Studentenpreis (Inland)	12 Ausgaben	€ 69,–
Studentenpreis (Ausland):	12 Ausgaben	€ 79,–

Einzelverkaufspreis:

Deutschland:	€ 7,50
Österreich:	€ 8,60
Schweiz:	sFr 15,80

Erscheinungsweise: monatlich