

„WIR BRAUCHEN EINE ZIELLINIE“

Jeff De Luca über die Wurzeln von FDD, den Charakter agiler Methoden und die Beziehungen zwischen FDD, XP und Scrum

Feature-Driven-Development (FDD) wird gemeinhin zu den verbreiteten agilen Methoden gezählt. Dabei nimmt FDD in mancherlei Hinsicht eine Sonderstellung ein, z. B. beim Thema Modellierung. Daher möchte manch einer FDD die Zugehörigkeit zur Gruppe der agilen Methoden absprechen. Jeff De Luca beantwortet Fragen zu den FDD-Wurzeln, Agilität und dem Verhältnis von FDD zu anderen agilen Methoden.

Was sind die Wurzeln von FDD und wann ist es entstanden?

Unter diesem Namen entstand FDD 1997/1998 in einem Projekt, das ich für eine Bank in Singapur durchführte. Ich hatte Peter Coad angeheuert, um die übergreifende Modellierung im Projekt zu leiten, so kamen wir beide zusammen. Die meisten Dinge im Singapur-Projekt hatte ich bereits seit Jahren so oder ähnlich gemacht und einiges davon ging zurück auf meine Zeit bei IBM. Peter Coad brachte dann die Idee sehr fein granularer Features ein, die sich als sehr wichtiges Konzept erwiesen haben. Sein Konzept beinhaltete kleinere Features, als ich sie bis dahin verwendet hatte. Ich würde also sagen, dass meine IBM-Erfahrungen zusammen mit den fein granularen Features von Peter Coad im Jahr 1997 die Wurzeln für FDD bildeten. Benannt und beschrieben haben wir FDD 1998.

Wie kamen Sie darauf, dass FDD einen eigenen Namen und eine eigene Methodenbeschreibung verdiente?

Nach einem gemeinsamen Essen mit John Cage, Chief Scientist bei Sun Microsystems, etwa in der Mitte des Projekts, ging ich zu Peter und wir vereinbarten, etwas über unseren Ansatz zu schreiben. Als ich mit John Cage essen war, hat er von seiner Teilnahme an Expertenkommissionen der Regierungen erzählt. Er war in Asien, um am APEC Meeting of Governments¹⁾ teilzunehmen. Auf jeden Fall berichtete er von einer bestimmten Expertenrunde, an der er teilgenommen hatte. Die Teilnehmer befanden sich in einem großen Raum und die Wände hatten Rollen und waren beweglich. Die Teams haben an den Wänden gearbeitet und zur Präsentation ihrer Ergebnisse die Wände in die Mitte des Raumes geschoben.

Das hört sich für Agilisten vertraut an.

Ich habe ihm dann unseren Ansatz zur Modellierung und Anforderungsanalyse beschrieben, bei dem die Teams mit

Anwendern zusammen an Wänden arbeiten. Dabei benutzen sie farbige Post-Its auf Flipchart-Papier und bringen ihre Ergebnisse in die Mitte des Raumes, um ihr Modell zu präsentieren. Außerdem habe ich Projektplanung und Tracking mit Diagrammen und Reports an der Wand durchgeführt (Wandplanung). John war begeistert und fragte mich: „Wer ist Euer Projekt-Anthropologe? Wer schreibt das Zeug auf?“. Das hat mich dazu bewegt, zu Peter zu gehen und aufzuschreiben, wie ich Software-Entwicklungsprojekte durchführe.

Agile vs. schwergewichtige Methoden

FDD wird zu den agilen Methoden gezählt. Was unterscheidet FDD von anderen agilen Methoden wie XP oder Scrum?

Es gibt eine ganze Menge von Unterschieden auf der Detailebene zwischen den agilen Methoden, aber ich finde diese nicht so interessant. Wirklich interessant ist das, was alle agilen Methoden miteinander verbindet: ihr gemeinsames Wertesystem²⁾. Dort steht z. B. „Lauffähiger Code hat Vorrang vor ausgedehnter Dokumentation“. Es ist ein sehr gutes Wertesystem und es braucht echte Reife, um es wirklich zu verstehen.

Es war ein langer Weg, bis agile Methoden akzeptiert wurden.

Noch vor wenigen Jahren gab es auf den großen Konferenzen regelmäßig Sessions, normalerweise gehalten von Business-Analysten, über die Unterschiede zwischen traditionellen, schwergewichtigen Methoden und agilen Methoden. In den Sessions gab es immer eine Folie überschrieben mit Titeln wie „Probleme mit Agil“, auf denen beispielsweise stand „braucht einen Kunden zur Zusammenarbeit“ oder „erfordert gute Leute“. Ganz im Ernst, kein Witz! Natürlich ist das richtig, aber das sind Probleme, die es bei jeder Art von Softwareentwicklung gibt. Oder sagen die Analysten im Endeffekt, dass man bei schwergewichtigen Methoden nicht mit dem Kunden reden muss und dass man dort keine guten



Jeff De Luca

lebt in Australien und arbeitet als Projektleiter, Berater und Autor. Er entwickelte die agile Methode Feature-Driven-Development (FDD) vor 10 Jahren. Er verließ IBM 1993, als er den Rang eines Senior Systemstrategen innehatte und gründete sein eigenes Beratungsunternehmen Nebulon, mit dem er seitdem weltweit Projekte durchführt. Seine bisher größte Herausforderung war das Singapur-Kreditprojekt 1995-1999.

Leute braucht? Ich glaube, wir wissen, wie erfolgreich so ein Projekt wäre. Das ist also Unsinn. Das sind Probleme, die bei allen Ansätzen zur Softwareentwicklung auftreten. Nur stellen die agilen Ansätze sie – im Gegensatz zu den traditionellen Methoden – ins Zentrum. Das ist das, was das Agile Manifest leistet.

FDD-Modellierung

Eine der auffälligen Unterschiede zwischen FDD und Scrum/XP ist der explizite Modellierungsprozess am Anfang. Einige würden sagen, das sei großer Entwurf am Anfang (Big Design Upfront, BDUF). Ist es das?

Nein, überhaupt nicht. BDUF ist eine abwertende Phrase geworden und einige scheinen jegliche Aktivität vor dem Programmieren BDUF zu nennen. Das ist nicht hilfreich. Als Projektmanager gehört es zu den Mühseligkeiten, dass man zwei Fragen beantworten muss. „Wie weit seid Ihr?“ und „Wie viel ist noch zu tun?“ Diese Fragen sind schwer zu beantworten. Sie sind noch schwerer mit Exaktheit zu beantworten und noch einmal schwerer mit Exaktheit und in einer Form, die für den Kunden verständlich ist. Der Prozess, ein Gesamtmodell zu entwickeln, ist weder schwergewichtig noch sehr detailliert. Das Ergebnis ist ein so genanntes Formmodell oder „Shape Model“. Das bedeutet, dass wir alle Klassen des Anwendungsbereichs und ihre Verbindungen identifizieren wollen, aber nicht alle Attribute und Methoden. Das geschieht in enger Ko-

²⁾ Anmerkung der Redaktion: Gemeint ist das „Agile Manifest“.

operation zwischen Entwicklern und Anwendern, die in FDD als Domänenexperten bezeichnet werden. Man kann sich das vielleicht als Grobdesign vorstellen, aber es ist auch Anforderungsanalyse. Während wir das Formmodell erstellen, ist das eigentliche Geheimnis der unglaubliche Wissenstransfer, der stattfindet. Modelle sind sehr aussagekräftig. Sie sind visuell explizit – es gibt wenig Interpretationsspielraum in einem Modell. Unser Verständnis des Anwendungsbereichs zu modellieren, indem wir mit den Anwendern sprechen, liefert uns viele Informationen und Wissen. Also ist der erste Prozess in FDD, das Entwickeln des Gesamtmodells, eine informationelle und analytische Tätigkeit, die uns das Wissen vermittelt, um eine Feature-Liste zu erstellen und dann das Projekt sehr genau und umfassend zu planen. Nachdem die Startphase abgeschlossen ist, sind die FDD-Iterationen und -Inkrementen in der Konstruktionsphase feingranularer als in typischen iterativen/ inkrementellen Methoden. Also, wie lange dauert diese so genannte Upfront-Aktivität in FDD? Bezogen auf sechs Monate Entwicklungszeit modellieren wir jeweils zwei Wochen lang mit den Anwendern. Jim Highsmith hat das in einem seiner Bücher gut zusammengefasst: In einem Sechs-Monatsprojekt kodiert Jeff De Luca in der dritten Woche, während Ron Jeffries (XP) in der ersten Woche kodiert; kaum ein signifikanter Unterschied.

Der Übergang vom Wasserfall-Ansatz zur iterativen und inkrementellen Entwicklung war ein großer Fortschritt. Iterative/inkrementelle Methoden nehmen eine Anzahl von Schnitten durch die meisten der Wasserfall-Phasen vor. Aber wenn wir zu puristisch in unserem iterativen/inkrementellen Ansatz sind, also z.B. auch Anforderungen und Analyse zerschneiden, dann ist es für uns natürlich auch schwer die Fragen „Wie weit seid Ihr?“ und „Wie viel ist noch zu tun?“ zu beantworten. Wir haben uns schließlich nicht einmal die restlichen Anforderungen angesehen. Also muss es zum Projektbeginn eine informationelle und analytische Aktivität geben, die uns das Wissen um die Ziellinie liefert, gegen die wir tracken und berichten, um diese Fragen zu beantworten. FDD ist die einzige agile Methode, die dies richtig macht. Es geht also nicht darum, „den ganzen Entwurf vorab machen“ (was BDUF abwertend bedeutet), sondern „gerade ausreichend viel Entwurf“. Dabei ist zu beachten, dass diese erste Aktivität in FDD genauso Anforderungsanalyse ist wie auch Grobdesign.

Sie sagen, der Haupteffekt des Prozesses „Entwickle Gesamtmodell“ sei es, den Anwendungsbereich und die Anforderungen zu verstehen. Das hört sich so ähnlich an wie das, was Eric Evans mit Domain Driven Design macht.

Wenn es dasselbe Ergebnis erbringt, ist es eine gute Idee. Wenn ich den Leuten zeige, wie man FDD abstrahiert und adaptiert, ist dieser erste FDD-Prozess eine der Stellen, an der man einen anderen Ansatz verwenden kann. Den Anwendungsbereich so zu modellieren, wie FDD es beschreibt, ist der beste Weg, den ich kenne. Wenn es aber ein anderer Ansatz schafft, uns mit einer informationellen und analytischen Aktivität so viel Wissen zu verschaffen, dass wir die Ziellinie sehr genau definieren können, dann ist das wunderbar für mich.

Chefarchitekten und -Projektmanager

Den Prozess „Entwickle Gesamtmodell“ haben Sie als eine gemeinsame Aufgabe verschiedener Projektbeteiligter beschrieben. Eine der FDD-Rollen ist der Chefarchitekt. Spielt er in diesem ersten Prozess eine besondere Rolle?

Die Rolle des Chefarchitekten wird überbewertet. Ich habe schon lange vor, die FDD-Prozesse zu überarbeiten, und dies ist eine Stelle, die sich ändern wird. Der korrekte Name der Rolle ist Chef-Modellierer (*Chief Modeler*) und so verwende ich ihn auch. Die wichtigen Rollen und Aspekte sind diese: Wir bringen die Fachexperten (Anwender, Analysten, Fachspezialisten) und Entwickler unter der Führung eines Moderators und eines erfahrenen Modellierers zusammen. Der Chef-Modellierer (oder Chefarchitekt in der aktuellen Prozessbeschreibung) ist einfach dieser erfahrene Modellierer, der die Gruppe soweit nötig führen kann, eine Modellskizze erstellt, um Fortschritt herzustellen, etc. Das sind alles nur Rollennamen. Rollen sind wie Hüte, die man aufsetzen kann, und man kann mehrere Hüte tragen. Ein Beispiel: Wenn eine Person sehr gut ist, kann sie den Hut des Moderators und den des Chef-Modellierers tragen. Peter Coad ist ein exzellentes Beispiel für so eine Person.

Das weicht von dem ab, was viele Leute intuitiv mit Chefarchitekt assoziieren.

Sie haben Recht. Es ist ein problematischer Name – alles mit „Architekt“ oder „Architektur“ im Namen ist in der IT problematisch, weil diese Begriffe so überladen sind.

Gibt es ein ähnliches Problem mit der Projektmanager-Rolle? Traditionell ist

der Projektmanager dafür verantwortlich, Aufgaben zu definieren und sie den Entwicklern zuzuweisen.

Die von Ihnen genannten Aufgaben werden in FDD anders gehandhabt. In der FDD-Beschreibung kommt der Projektmanager kaum vor – nur an drei oder vier Stellen – und normalerweise in der Rolle „stellt das Team zusammen, das die nächsten Aufgaben oder Aktivitäten durchführt“. Die Arbeitsaufgaben in einem FDD-Projekt, d.h. die Features, werden nicht vom Projektmanager definiert. Die Grobplanung machen Projektmanager und Entwickler gemeinsam. Die Zuweisung von Features für Entwurf und Programmierung nehmen Projektmanager und Entwickler gemeinsam vor. Die konkrete zeitliche Planung und die Gruppierung von Features für Entwurf und Programmierung machen die Entwickler selbst.

FDD-Features

Der Prozess „Entwickle Gesamtmodell“ kommt vor dem Prozess „Erstelle eine Feature-Liste“. Man könnte argumentieren, die Reihenfolge sei falsch. Muss man nicht die konkreten Anforderungen und Features kennen, um das Gesamtmodell zu entwerfen?

Nein, nicht wie diese Dinge in FDD definiert sind. Ein FDD-Feature ist ein winzig kleiner Teil einer für den Kunden nützlichen Funktion. Wenn man die Feature-Liste erstellt, möchte man möglichst kleine Features haben, weil einem das die höchste Flexibilität bei der detaillierten zeitlichen Planung der Features während der Entwicklungsphase gibt. Um den Anwendungsbereich in eine Liste feingranularer Features mit guter Abdeckung und hoher Genauigkeit aufzubrechen, braucht man vorher eine informationelle und analytische Aktivität, die einen mit dem notwendigen Wissen versorgt. Das Singapur-Darlehensprojekt hatte über 1.000 Features nur in der Fachlogik-Schicht. Mit diesem Verständnis eines FDD-Features ist es natürlich unmöglich, die Feature-Liste zuerst zu erstellen. Der andere Teil der Frage zielt auf das ab, was man wissen muss, um das Gesamtmodell zu erstellen. Wir modellieren den Teil der Anwendungsdomäne, der durch den Umfang des Projektes oder der Anwendung bestimmt wird. Vor der Entwicklung des Gesamtmodells müssen wir also mindestens eine Idee davon haben, welcher Teil der Anwendungsdomäne zu betrachten ist. Ohne das gibt es keine Grenze, gegen die man arbeiten kann, und man kann die ganze Welt modellieren. Es ist schwer, das Ziel zu treffen, wenn man nicht

weiß, was das Ziel ist. So macht man in FDD das Ziel klar. In der IT sind wir in diesem Bereich sehr schwach; wir machen zu viele Annahmen und definieren das Ziel nicht klar.

Class-Ownership

XP und andere Ansätze benutzen Collective Code-Ownership, während FDD Class-Ownership³⁾ praktiziert. Was ist der Grund dafür? Wie strikt wird Class-Ownership in typischen FDD-Projekten gehandhabt?

Das Konzept des Modul- oder Komponentenbesitzes wird seit langem praktiziert und ist als „Best Practice“ bekannt. So wird jede professionelle Entwicklung im Großen durchgeführt. Es kann nicht jeder über alles Bescheid wissen. Brooks („The Mythical Man Month“) hat uns das vor annähernd 30 Jahren gesagt und viele andere ebenfalls. Aus irgendwelchen Gründen ist dieses Konzept aus einigen Methoden verschwunden und wir müssen es zurückbringen. Denken Sie an ein Sequenzdiagramm für ein Modell: Einige Klassen kooperieren, um eine Funktion auszuführen, z.B. die Gewichtsrechnung einer Fracht im Anwendungsbereich Produktverkauf. Vier oder fünf Klassen sind an einem solchen Feature beteiligt. Wenn man keine sichere Softwareentwicklung praktiziert, wenn man kein Class-Ownership betreibt und die Gewichtsrechnung an einen Entwickler gibt, muss dieser vier oder fünf Klassen im System ändern und diese kennen. Jetzt stellen wir uns eine Klasse vor, die sehr häufig im System benutzt wird, wie z.B. die Klasse „Kreditantrag“ in einem Kreditsystem. Das ist eine Klasse mit hunderten von Methoden und Attributen, die alle von verschiedenen Entwicklern zu unterschiedlichen Zeiten geschrieben wurden. Da sträuben sich mir die Nackenhaare. Und besonders ironisch ist, dass eines der fundamentalen OO-Prinzipien Kapselung ist; was eine Klasse tut, ist privat, und die Implementierungsdetails können sehr stark variieren, solange die Klasse ein konsistentes Interface zum Rest der Welt hat. Menschen kapseln naturgemäß. Wenn man Class-Ownership praktiziert, bekommt man eine viel höhere Konsistenz der Implementierung und des Interfaces. Class Ownership gilt nicht lebenslanglich. Wenn man für die Kundenklasse verantwortlich ist, muss das nicht für immer so bleiben. Die Klassenbesitzer können und werden sich über die Projektlaufzeit ändern. Letztlich sagen wir nur, dass ein Entwickler einige Klassen besitzt. Wenn es sich um ein kleines Projekt handelt, können wir es „auslosen“,

z.B.: „Fred, Du nimmst den Kunden; Paul, Du kümmerst Dich um das Kontenzeug; Alex, Du machst die Administrationsdinge; Phil ist für die Benutzungsoberfläche zuständig.“ Wenn es sich um ein größeres Projekt handelt mit einem größeren oder komplexeren Modell, dann denken wir intensiver darüber nach. Wir sehen uns die Klassen im Modell an und denken über die Klassenkomplexitäten und die Entwicklerfähigkeiten nach. Man kann zwei Arten von Klassenkomplexität unterscheiden: durchdringende Komplexität (d. h. eine Klasse ist an vielen Features beteiligt, tut aber selbst nicht viel, außer Delegation) und algorithmische Komplexität. Also denken wir in einem größeren Modell intensiver darüber nach. Wir wollen nicht einem Entwickler viele durchdringend komplexe und algorithmisch komplexe Klassen zuweisen, weil das zu dessen Blockade während der Programmierung führt.

Testen und Code-Inspektionen

Testen wird von vielen agilen Methoden hervorgehoben. FDD benutzt Code-Inspektionen. Welche Rolle spielt Testen in FDD?

Zunächst muss man die anderen Methoden fragen, welches Ziel sie mit Testen verfolgen. Die Antwort ist: „Fehler beseitigen“. Dieses Ziel wird in FDD sogar noch stärker betont, aber nicht nur durch Testen, weil Testen einer der am wenigsten effektivsten Wege zur Fehlerbeseitigung ist. Das ist nicht spekulativ, sondern in der IT sehr gut untersucht und gemessen (z. B. von Capers Jones). Inspektionen sind die effektivste Form der Fehlerbeseitigung und darüber hinaus eine sehr gute Teambildungsaktivität, weil sie die Teamkultur aktiv verbreiten genauso wie syntaktische und semantische Standards. FDD nutzt beim Testen bewährte „Best Practices“, um möglichst wenig Fehler auszuliefern. Wir haben das kooperative Modellieren mit den Anwendern, wo wir eine genaue Anforderungsanalyse durchführen, weil Modelle so visuell-explizit sind. Dann machen wir formelle Entwurfs- und Code-Inspektionen, aber nur für fein-granulare Features, was die Umsetzung dieser Best Practices viel einfacher macht. All das tun wir, um möglichst wenig Fehler haben, bevor wir mit den Unit-Tests beginnen, die ein vorgeschriebener Schritt im Prozess „Konstruiere je Feature“ sind. Wenn wir dort angekommen sind, testen wir genauso viel wie alle anderen Testansätze.

Management

Was mögen Manager und Anwender an FDD?

Manager lieben das Tracking und Reporting von FDD, die Transparenz und Kommunizierbarkeit. Erledigte Arbeit in kundenverständlichen Features zu beschreiben und das Reporting gegen die Dekomposition des Anwendungsbereiches selbst zu fahren, harmoniert sehr gut mit der Denkweise der meisten Branchen. Die Featureliste und das Parking-Lot-Diagramm (eine Reporting-Visualisierung in FDD) haben eine weite Verbreitung gefunden. Viele andere Methoden nutzen diese Dinge heute ebenfalls. Projektmanager wollen die folgenden zwei Fragen beantworten können: „Wie weit seid Ihr?“ und „Wie viel ist noch zu tun?“ Die Antwort auf diese Fragen wollen sie für den Kunden verständlich kommunizieren. Das macht FDD.

FDD und andere agile Methoden

Es ist durchaus üblich, verschiedene agile Ansätze in Projekten zu mischen, z.B. Scrum-Meetings zu haben und Pair-Programmieren aus XP. Sind Techniken aus Scrum und XP auch in FDD-Projekten anzutreffen?

Ja, das sind sie. Viele der 12 XP-Techniken sind natürlich auch in FDD vorhanden – sie sind nicht neu. Tägliche Standup-Meetings verwende ich seit Jahrzehnten. Tatsächlich haben wir diese schon 1997 während der frühen Phasen des Kreditprojektes in Singapur verwendet. Das Burndown-Chart aus Scrum ist ähnlich zum Trend-Chart in FDD (die einen zeichnen eine fallende Linie, die anderen eine steigende). Auf dieser Ebene gibt es viele Ähnlichkeiten und Unterschiede. FDD benutzt kein Timeboxing für das Management wie Scrum. FDD benutzt kein Pair-Programmieren. Aber die mit den Techniken verbundenen Ziele werden auch mit FDD erreicht, aber auf anderen Wegen.

Projekttypen

Sie haben gesagt, dass Collective Ownership nicht gut skaliert. Wie groß sind typische FDD-Projekte? Was ist das größte FDD-Projekt, das Sie je durchgeführt haben oder kennen?

Das Kreditprojekt in Singapur war eines der größten Java-Projekte seiner Zeit: Es hatte ca. 50 Mitarbeiter. Das größte Entwicklerteam, das ich jemals in einem Projekt hatte, waren 250 Entwickler. Im Kleinen habe ich viele Zwei-Personen-FDD-Projekte durchgeführt. Die meisten Projekte heutzutage sind kürzer (die meisten Unternehmen sind nicht interes-

³⁾ Anm. der Redaktion: Klassen werden an bestimmte Entwickler gebunden.

siert an Projekten, die länger als ein Jahr dauern) und das reduziert auch die Teamgrößen. Ich würde sagen, die meisten nennenswerten Projekte haben zwischen sechs und 50 Entwicklern. Natürlich gibt es auch nennenswerte Projekte mit mehr als 50 Entwicklern, aber das ist nicht der Regelfall. Ich bin außerdem meistens mit Geschäftsanwendungen betraut. Ich sage nicht nur, dass Collective Ownership nicht gut skaliert, sondern ich glaube auch, dass es nicht so gut funktioniert wie andere Ansätze. Es ist nicht bloß eine Frage der Skalierung.

Würden Sie sagen, dass FDD für bestimmte Projekttypen besser geeignet ist als für andere?

Nein, ich glaube nicht, dass sich agile Methoden nach dem Projekttyp klassifizieren

lassen. Ich rate den Leuten immer, sich Jim Highsmiths Buch „Agile Software Development Ecosystems“ zu besorgen. Das ist das einzige Buch, das alle agilen Methoden mit ausreichender Detaillierung diskutiert. Man liest das Buch und ein oder zwei Methoden fühlen sich gut für einen selbst und das eigene Team an. Diese sollte man sich dann näher ansehen. Ich würde daher sagen, dass agile Methoden eher geeignet sind für bestimmte Typen von Personen und Unternehmenskulturen als für Projekttypen.

Die Zukunft von FDD

Arbeiten Sie zurzeit an speziellen Verbesserungen von FDD?

Nein, nicht an Verbesserungen an FDD als solches. Aber ich beschreibe und erkläre FDD und

auch die Beziehungen zu anderen IT-Aspekten, wie Programmmanagement, Governance etc.

Jeff, vielen Dank für das Gespräch.

Das Interview führte Stefan Roock, Senior IT-Berater bei der akquinet AG in Hamburg, auf Englisch.

Links

- Das agile Manifest: agilemanifesto.org
- Kurze englischsprachige FDD-Definition: www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf
- Community-Website zu FDD: www.featuredrivenddevelopment.com
- Deutschsprachige FDD-Infos und FDD-Trainings in Deutschland: fdd.it-agile.de
- Website von Jeff De Luca: www.nebulon.com