Ein Plädoyer für Datenflussdiagramme aus der Sicht der Aufwandsschätzung und der agilen Softwareentwicklung

Bernhard Daubner, Andreas Henrich

Lehrstuhl für Angewandte Informatik I, Universität Bayreuth
bernhard.daubner@uni-bayreuth.de, andreas.henrich@uni-bayreuth.de

Abstract: Die mit der Strukturierten Analyse von DeMarco bekannt gewordenen Datenflussdiagramme zur Modellierung von Softwaresystemen sind mit zunehmender Verbreitung objektorientierter Modellierungstechniken in Vergessenheit geraten. Wir wollen zeigen, dass dieser Diagrammtyp aus der Sicht der Aufwandsschätzung und der agilen Softwareentwicklung Vorteile gegenüber den üblicherweise in der objektorientierten Analyse verwendeten Modellierungstechniken aufweist und auch heute noch sinnvoll eingesetzt werden kann.

1 Einführung und Motivation

1.1 Agile Softwareentwicklung

In heutigen IT-Projekten wird Software in kurzen Zyklen und mit ständig wachsender Funktionalität entwickelt. Dem ist das klassische Projektmanagement, das die Ziele des Entwicklungsprozesses von Anfang an festlegt, nicht gewachsen. Die Anforderungen an ein zu entwickelndes oder zu implementierendes IT-System können vom Kunden anfangs nur ungenau spezifiziert werden und ändern sich dann im Laufe des Entwicklungsprozesses aufgrund neuer Erkenntnisse oder äußerer Einflüsse.
Als Reaktion darauf sind in den letzten Jahren mehrere leichtgewichtige Softwareentwicklungsprozesse, welche sich auch als "agil" bezeichnen (vgl. [Co02]), entstanden. Der wohl bekannteste Vertreter dieser leichtgewichtigen Entwicklungsmethoden ist das *eXtreme Programming* nach Beck [Be99]. Diese Modelle zeichnen sich dadurch aus, dass unter permanenter Miteinbeziehung des Kunden durch iteratives Weiterentwickeln der Software mit extrem kurzen Release-Zyklen die gewünschte Funktionalität implementiert wird.

1.2 Überlegungen zur Vertragsgestaltung

Allerdings kann bei derartigen iterativen Softwareentwicklungsmodellen der Gesamtumfang des Softwareentwicklungsprojektes und damit die Projektdauer sowie der Erstellungsaufwand nur schwer abgeschätzt werden. Denn wenn die Anforderungen an das zu entwickelnde System iterativ weiterentwickelt oder sogar im Laufe des Projektes überhaupt erst definiert werden, kann zu Projektbeginn kein entsprechendes Verfahren zur

Abschätzung des Projektumfangs angewendet werden. Die meisten Kunden, die ein IT-Projekt in Auftrag geben, werden jedoch Aussagen über den Projektumfang und die damit verbundenen Projektkosten als Vertragsbestandteil fordern.
Beck und Cleal [BC99] beschreiben in diesem Zusammenhang so genannte *Optional Scope Contracts* (Verträge mit wahlfreiem Umfang) als mögliche Alternative. Die Grundidee dabei ist, dass vertraglich nur die Leistungserbringer, der Tätigkeitszeitraum und das Honorar festgelegt werden. Der genaue Umfang der zu erbringenden Leistung wird nicht definiert. Allerdings wird von Seiten des Dienstleisters versprochen, nach seinem Vermögen die bestmögliche Leistung zu erbringen.
Eine derartige Vorgehensweise wird von Müller in [Mü03] als *Iteratives Festpreismodell* bezeichnet, da die Beteiligten jeweils für die nächste Stufe des Projektablaufes einen Festpreis vereinbaren und für die weiteren Schritte Vorhersagen abgeben. Jede vergangene Iteration fungiert dabei als Basis für den Abgleich von Erwartungen und geleisteter Arbeit und beeinflusst die zukünftige Kalkulation.
Lott schränkt in [Lo97] die Ausgestaltungsmöglichkeiten derartiger *Staged Contracts* (Phasenorientierte Verträge) dahingehend ein, dass sich der jeweilige Vertragsumfang an signifikanten Meilensteinen des Entwicklungsprozesses orientieren soll. So könnte ein erster Vertrag zur Projektfindung abgeschlossen werden, in der eine Ist-Analyse durchgeführt und die Anforderungen des Kunden ermittelt werden. Ein zweiter Vertrag würde dann den Analyse- und Design-Aktivitäten gewidmet sein, während ein dritter Vertrag die Konstruktion der Software und deren Inbetriebnahme beschreibt.
Der Kerngedanke dieser Vertragsausgestaltungen ist, dass der Dienstleister zu Vertragsbeginn nicht gezwungen ist, irgendwelche Angaben über Umfang und Dauer des Projektes zu machen, für deren Einschätzung er zu diesem Zeitpunkt noch nicht die notwendigen Informationen hat. Zusätzlich behalten beide Vertragspartner sich das Recht vor, nach dem aktuellen Vertrag keinen Folgevertrag abzuschließen, weil z. B. der Auftraggeber mit der erbrachten Leistung nicht zufrieden war oder aber der Dienstleister evtl. festgestellt hat, dass eine Fortführung des Projektes seine personellen Möglichkeiten übersteigt.

1.3 Leichtgewichtige Aufwandsschätzung

Üblicherweise werden zu Beginn des Softwareentwicklungsprozesses die Geschäftsprozesse des Unternehmens und insbesondere die Anwendungsfälle erarbeitet, die durch die zu entwickelnde Software abgedeckt werden sollen. Bei objektorientierten Vorgehensmodellen werden diese Anwendungsfälle mit Hilfe von UML Use Case Diagrammen beschrieben. Zusätzlich werden UML Aktivitätsdiagramme eingesetzt, um die Abläufe von Geschäftsprozessen zu beschreiben. Beim eXtreme Programming werden an dieser Stelle die Anforderungen zusammen mit dem Kunden als so genannte *Customer Stories* ermittelt.
Wir sind jedoch der Meinung, dass es im Sinne einer Strukturierung und Präzisierung der Überlegungen vorteilhaft ist, zusammen mit den Anwendungsfällen auch die Funktion des zu entwickelnden Systems mit Hilfe der auftretenden Datenflüsse zu beschreiben. Dass dies auch eine frühzeitige Aufwandsschätzung erlaubt, ist ein positiver Nebeneffekt. Dies lässt sich mit den Methoden der *Strukturierten Analyse* (vgl. [De79] und [Yo89])

und insbesondere durch die *ereignisorientierte Zerlegung* nach McMenamin und Palmer [MP88] erreichen. Dabei wird das System in so genannte *essentielle Aktivitäten* als Reaktion auf bestimmte Ereignisse aufgeteilt. Mit der Kenntnis der bei den essentiellen Aktivitäten auftretenden Datenflüsse ist bereits eine erste Aufwandsschätzung durch die Function Point Analyse (PFA) (vgl. [BPS91]) möglich, da die Datenflussmodelle die wesentlichen Informationen über die bei der FPA bewerteten Transaktions- und Datenkomponenten des untersuchten Software-Systems enthalten (vgl. Abb. 1).

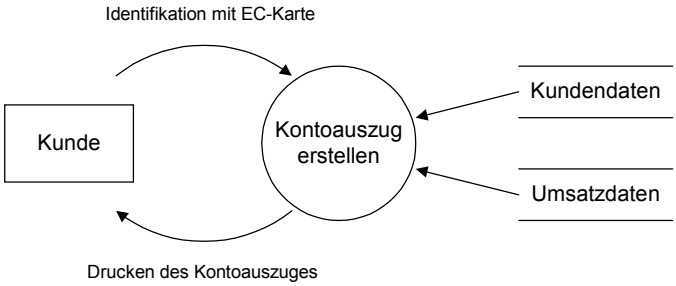

Abb. 1: Funktionsweise eines Kontoauszugsdruckers

2 Integration funktionaler und objektorientierter Modellierung

2.1 Funktionale versus objektorientierte Modellierung

Es soll hier allerdings nicht der Eindruck erweckt werden, dass wir die objektorientierte Modellierung mit der Rückbesinnung auf die strukturierten Methoden und die funktionale Analyse in Frage stellen wollen. Wir sind vielmehr der Meinung, dass insbesondere in der Analysephase die objektorientierten Techniken durch die funktionale Analyse ergänzt werden können, was auch von Rumbaugh et. al. (vgl. [Ru91], S. 123) so dargestellt wurde.
Coad und Yourdon [CY91] sehen als die größten Nachteile der strukturierten Methoden, dass zum einen die Beschreibung der Datenstrukturen vernachlässigt werde, da die Betonung auf der Funktionalität liege. Zusätzlich gebe es Probleme beim Übergang von der Analyse zum Design durch den Wechsel der grundlegenden Darstellungsform. So würden in der Analysephase Datenflussdiagramme (DFD) zur Darstellung der Prozesse, Terminatoren und Speicher verwendet werden, während in der Designphase ein Programm als hierarchische Anordnung von Modulen mittels so genannter *Structure Charts* dargestellt werde.
Dagegen werden von Scheer [Sc90] die Datenflussdiagramme als Bindeglied zwischen der Beschreibung der Datenstrukturen im Datenmodell und der Beschreibung von Vorgängen im Geschäftsprozessmodell bewertet. Eine Einschätzung, die wir teilen.
Außerdem ist man durch die Verwendung von Datenflussdiagrammen in der Analysephase nicht von vornherein auch auf die strukturierten Design-Methoden festgelegt. Es lässt sich sogar zeigen, dass sich aus den DFD Kandidaten für die Objekte und Klassen

der objektorientierten Analyse ableiten lassen, die dann noch weiter verfeinert werden müssen. Sully beschreibt in [Su93], dass man aus Datenflussdiagrammen Objekte und Objektstrukturen ableiten könne, indem man die Datenspeicher als Kerne von Objekten auffasse, um die sich Teile der auf sie zugreifenden Funktionen als Methoden zu einer Schale anlagern ließen.

2.2 Datenflussmodellierung mit UML

Um jedoch die bei der Analyse der Datenflüsse erstellten Diagramme soweit wie möglich wieder verwenden und daraus im Laufe des Entwicklungsprozesses andere benötigte Diagramme ableiten zu können, schlagen wir vor, die Datenflussdiagramme mit den Darstellungsmöglichkeiten der UML zu modellieren. Die klassischen DFD nach DeMarco bestehen aus Terminatoren, Prozessen, Speichern und Datenflüssen. Die Prozesse können dabei unmittelbar als UML Aktivität dargestellt werden. Statt der Datenspeicher und der Terminatoren stellen wir die entsprechenden Geschäftsobjekte dar. Dabei beschreibt ein Geschäftsobjekt einen Gegenstand, ein Konzept, einen Ort oder eine Person aus dem realen Geschäftsleben in einem Detaillierungsgrad, wie er auch von Fachabteilungen und damit vom Kunden verstanden werden kann (vgl. [Oes98]). Zusätzlich verwenden wir Stereotypen um Geschäftsobjekte, die in einem Datenspeicher abgelegt werden, von den als Terminatoren fungierenden Geschäftsobjekten zu unterscheiden (vgl. Abb. 2).

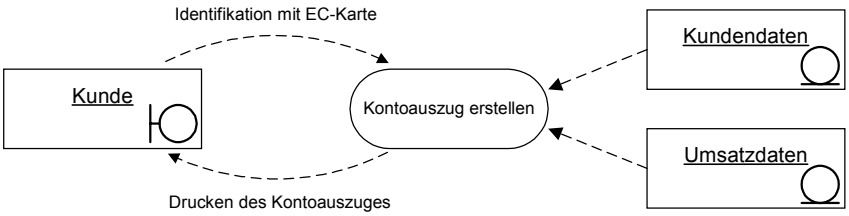

Abb. 2: in UML modellierte Datenflüsse

Damit sind bereits alle für die FPA benötigten Transaktions- und Daten-Komponenten (externe Dateneingaben, -ausgaben, -abfragen, interne und externe Datenbestände) zumindest quantitativ bekannt und können näherungsweise bewertet werden. Um zusätzlich die Komplexität der Datenflüsse bewerten zu können, müssten die Geschäftsobjekte noch detaillierter modelliert werden.
Auf diese Weise entsteht ein Aktivitätsdiagramm, welches nur Aktivitäten und Objekte, aber keine Bedingungs-, Synchronisations- oder Split-Elemente enthält. Man kann es nun bei der Erstellung dieser Diagramme bewenden lassen, wenn sie nur zur Modellierung der Customer Stories und zu einer leichtgewichtigen Aufwandsschätzung innerhalb der agilen Softwareentwicklung verwendet werden sollen. Grundsätzlich können diese Diagramme jedoch auch zu Ablaufdiagrammen mit entsprechenden Verzweigungen weiterentwickelt werden, um bestimmte Prozesse detaillierter zu beschreiben. Da ein Aktivitätsdiagramm per definitionem ein spezielles Zustandsdiagramm ist, könnte man auch

noch Zustände bzw. Ereignisse ergänzen, durch die die einzelnen Aktivitäten ausgelöst werden. Dadurch erhält man ein EPK-Diagramm, welches eine ereignisorientierte Prozesskette (vgl. [St01]) beschreibt.

3 Werkzeug-Integration

Eine fundierte Aufwandsschätzung nach der Function Point Methode stellt zunächst einmal eine Mehrarbeit für den Softwareentwickler dar. Insbesondere scheint sie nur schwer mit agilen Entwicklungsprozessen vereinbar, in denen der Fokus aller Aktivitäten auf der kontinuierlichen Weiterentwicklung der Software liegen soll. Das Ziel muss daher sein, die Aufwandsschätzung auf Basis von DFD zu automatisieren und in die Entwicklungsumgebung zu integrieren.
Wir wollen daher die Integrierte Entwicklungsumgebung *Together ControlCenter* dahingehend erweitern, dass damit die von uns beschriebenen UML-Datenflussdiagramme modelliert werden können. Dann soll Together um Werkzeuge ergänzt werden, mit denen eine möglichst automatisierte Bewertung dieser Modelle mittels FPA durchgeführt werden kann. Diese Bewertung soll auch inkrementell möglich sein, so dass die Ergebnisse mit zunehmenden Informationen über das Softwareentwicklungsprojekt immer genauer werden.

Literaturverzeichnis

[BC99] Beck, K.; Cleal, D.: Optional Scope Contracts. http://www.xprogramming.com, 1999.

[Be99] Beck, K.: Extreme Programming explained: Embrace Change. Addison Wesley, Upper Saddle River, New Jersey, 1999.

[BPS91] Bundschuh, M.; Peetz, W.; Siska, R.: Aufwandsschätzung von DV-Projekten mit der Function-Point-Methode. Verlag TÜV Rheinland, Köln, 1991.

[Co02] Alistair Cockburn: Agile Software Development. Addison-Wesley, Boston, MA, 2002.

[CY91] Coad, P.; Yourdon, E.: Object oriented analysis. Yourdon Press, Englewood Cliffs, New Jersey, second edition, 1991.

[De79] DeMarco, T.: Structured analysis and system specification. Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[Lo97] Lott, M. Ch.: Breathing new life into the waterfall model. IEEE Software, 14(5):103–105, 1997.

[Mü03] Müller, F.: Agile Softwareentwicklung. iX - Magazin für professionelle Informationstechnik, 2:109–111, 2003.

[MP88] McMenamin, St.; Palmer, J. F.: Strukturierte Systemanalyse. Hanser, München, Wien, 1988.

[Oe98] Oestereich, B.: Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language. R. Oldenbourg, München, fourth edition, 1998.

[Ru91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[Sc90] Scheer, A.-W.: EDV-orientierte Betriebswirtschaftslehre. Springer, Berlin, fourth edition, 1990.

[St01] Staud, J.: Geschäftsprozessanalyse: Ereignisgesteuerte Prozessketten und objektorientierte Geschäftsprozessmodellierung für Betriebswirtschaftliche Standardsoftware. Springer, Berlin, second edition, 2001.

[Su93] Sully, P.: Modelling the world with objects. Prentice Hall, New York et. al., 1993.

[Yo89] Yourdon, E.: Modern structured analysis. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.