Tom DeMarco
Strukturierte Analyse und System Spezifikation

Adrian Immanuel Kieß, Zweitsemester

Im schönen Sommer des Jahres 2004

Proseminar Wegweisende Arbeiten der Softwaretechnik
Universität Leipzig
Lehrstuhl für angewandte Telematik / e-Business

Prof. Dr. Volker Gruhn & Dipl.-Math. Ralf Laue

mailto:adrian@immanuelK.net
http://www.immanuelK.net

Inhaltsverzeichnis

[1] Tom DeMarco hat sich in *"Peopleware"* weiter mit menschlichem Verhalten in Entwicklungsprojekten auseinandergesetzt. Eine Rezension finden Sie unter: http://books.slashdot.org/article.pl?sid=98/09/15/1618241

1 Einklang

> "Einen Knoten?" fragte Alice hilfsbereit. "Soll ich dir helfen, ihn zu lösen?"
>
> "Alice im Wunderland", Lewis Caroll

1.1 Definition

Eine der sehr häufig angewandten Analysemethoden ist die *Strukturierte Analyse.* Tom DeMarco[2], der die Strukturierte Analyse als erster erfolgreich vertrat, definierte den Begriff *"Analyse"* folgendermaßen:

Abbildung 1: Tom DeMarco

> "Analysis is the study of a problem, prior to taking some action. In the specific domain of computer systems development, analysis refers to the study of some business area or application, usually leading to the specification of a new system."

Alternativ können wir auch definieren:

> Die *Analyse* ist ein Prozess, bei der wir uns von der realen Welt ein idealisiertes Modell erstellen und hierbei diverse Zwänge ignorieren.

[2]Tom DeMarco's Homepage: http://systemsguild.com/GuildSite/TDM/TDMBio.html. Hier finden Sie eine Bibliographie sowie eine komplette Publikationsliste.

1.2 Analyse und Design

Im Gegensatz zu der Analyse ist *Design* ein Prozess, in dessen Fortschreiten aus dem soeben gewonnenen Modell eine Implementierung erstellt wird. Während dieses Prozesses müssen wir das in der Analysephase erstellte Modell den gegebenen Zwängen anpassen.

Es ist wichtig, zwischen Analyse und Design prinzipiell bewusst zu unterscheiden. Die Analyse, angewandt auf einen "Heimcomputer", könnte ergeben, dass die Anforderungen an den Heimcomputer mit einem unendlich großen Speicherplatz und unenedlich schneller Prozessorleistung, bei keinerlei Stromverbrauch definiert werden. Die Designphase aber wird anschliessend diese Anforderungen durch gegebene Zwänge, wie zum Beispiel materielle, einschränken.

1.3 Die Analysephase

In der Analysephase muss das Problem genau untersucht werden, um alle notwendigen Kenntnisse über das Problemumfeld (Domain Knowledge) zu erhalten. Das Ergebnis der Analyse ist ein Analysemodell, welches eine Abstraktion des Problemes darstellt. Dieses Modell wiederum ist nun die Spezifikation des Systemes, welche keinen Implementierungsbedingungen unterlegen ist.

Um diese Aufgabe erfolgreich zu meistern, muss der Systemanalytiker nach DeMarco folgende Ergebnisse anstreben:

- Das optimale Ziel erkennen.
- Eine detaillierte Spezifikation des Zieles erstellen, damit eine Überprüfung auf die erfolgreiche Implementierung stattfinden kann.
- Eine Schätzung aller essenziellen Parameter, welche mit dem Ziel verknüpft sind. Hierbei kann es sich zum Beispiel um Kosten, Zeitpläne und Nutzen handeln.

Alle drei Punkte müssen bei den partizipierenden Gruppen Übereinstimmung finden.

1.3.1 Kommunikation

Um ein Softwareprojekt erfolgreich durchzuführen, ist eine allen Partnen verständliche Sprache erforderlich. Die Benutzer des zu erstellenden Systemes verwenden eine Fachsprache, welche für einen Fachbereich gängig ist. Auch Programmierer besitzen eine eigene Fachsprache, es kann daher leicht zu Missverständnissen kommen. Gleiche Worte können von beiden Gruppen als unterschiedliche Ideen und Konzepte verstanden werden. Die Aufgabe des Systemanalytikers ist es, die Anforderungen der Benutzer in fachlich-technische Spezifikationen für die Programmierer zu übersetzen.

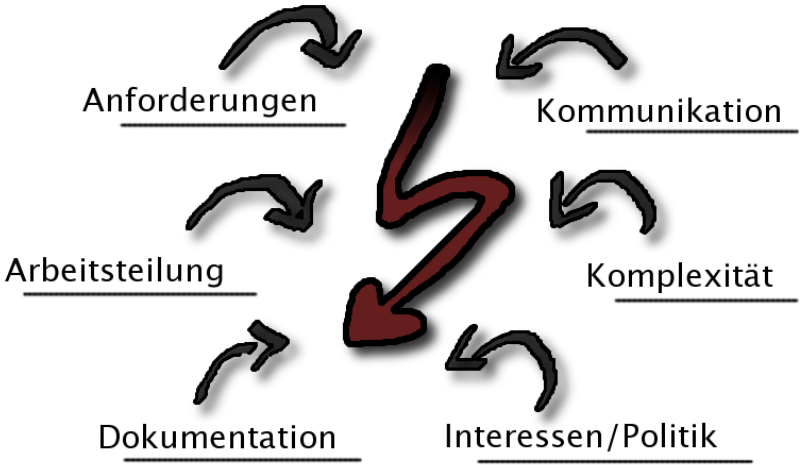
1.3.2 Sich ständig ändernde Anforderungen an das System

Während der Analysephase muss sich der Systemanalytiker in die Welt der Benutzer einarbeiten und deren Fachsprache erlernen. Es besteht die Möglichkeit, dass sich während der Projektlaufzeit Änderungen der Ziele oder Anforderungen der Umwelt ergeben, welche berücksichtigt werden müssen. Die Möglichkeit zu Anpassungen muss explizit mit einbezogen werden.

1.3.3 Dokumentation des Zieles

Eine Spezifikation für komplexe Systeme muss in viele kleine Spezifikationen zerlegt werden. Findet keine Zerlegung statt, werden die Spezifikationen schwer verständlich und unübersichtlich. Bei der Zerlegung

Abbildung 2: Das Problemumfeld



der Systeme in Subsysteme muss auf eine genaue Abgrenzung der Teilsysteme, sowie auf eine exakte Definition aller Schnittstellen Rücksicht genommen werden.

1.3.4 Politik[3]

Ein neues System bringt Unsicherheiten mit sich. Die Furcht vor Veränderungen von bestehenden Verhältnissen kann zu einer Verweigerung von Kooperation mit dem Systemanalytiker führen. Diese Tatsache darf *keinesfalls* unterschätzt werden.

2 Strukturierte Analyse

Bei der Strukturierten Analyse handelt es sich um eine Methode, welche hauptsächlich während der Analysephase eingesetzt wird. Die Strukturierte Analyse fußt im Wesentlichen auf drei, miteinander verknüpften Konzepten:

1. Datenflussdiagramme (Data Flow Diagrams)
2. Datenlexikas (Data Dictionaries)
3. Primitive Prozessspezifikationen (MiniSpecs)

[3]Tom DeMarco hat sich in *"Peopleware"* weiter mit menschlichem Verhalten in Entwicklungsprojekten auseinandergesetzt. Eine Rezension finden Sie unter: http://books.slashdot.org/article.pl?sid=98/09/15/1618241

Die Idee der Srukturierten Analyse hat sich mit der Veröffentlichung des Standartwerkes “Structured Analysis and System Specification” von Tom DeMarco Mitte der siebziger Jahre zu einem weit verbreiteten Standard bei Systemanalysen entwickelt und wird in vielen Software Entwicklungsumgebungen (CASE[4]) eingesetzt. Die Stärken der Strukturierten Analyse liegen vor allen Dingen darin, dass sie sich graphischer Objekte bedient und eine einfache Modellnotation besitzt. Eine Spezifikation der Anwendung in freiem, schwer verständlichem, zusammenhängenden Text wird vermieden.

2.1 Datenflussdiagramme

Software interagiert in jedem nur erdenklichen Einsatzbereich mit Daten. Daher besteht die hauptsächliche Aufgabe von Software darin, Daten, die eingegeben werden zu bearbeiten und dann wieder auszugeben. Die Eingabe einer Information in ein Softwaresystem kann zu vielen verschiedenen Ausgaben führen. Möchten wir an einem Geldautomaten Geld abheben, können wir Meldungen erhalten wie: “Der Betrag wird jetzt ausgezahlt”, oder aber Fehlermeldungen in der Art: “Ausser Betrieb” oder “Error 3xBF ocured”. Datenflussmodelle (Data Flow Diagrams) versuchen nun, den Weg dieser Informationen nachzumodellieren.

2.1.1 Anforderungen

Erarbeiten wir ein Datenflussdiagramm, so müssen wir uns die folgenden Fragen stellen:

- Welche Funktion soll das System erfüllen?
- Wie sollen die Funktionen interagieren?
- Was für Daten soll das System transferieren?
- Welche Eingaben werden zu welchen Ausgaben geführt?
- Was für grundlegende Arbeiten soll das System verrichten?
- Woher bezieht das System seine Informationen?
- Wohin werden die Ergebnisse fließen?

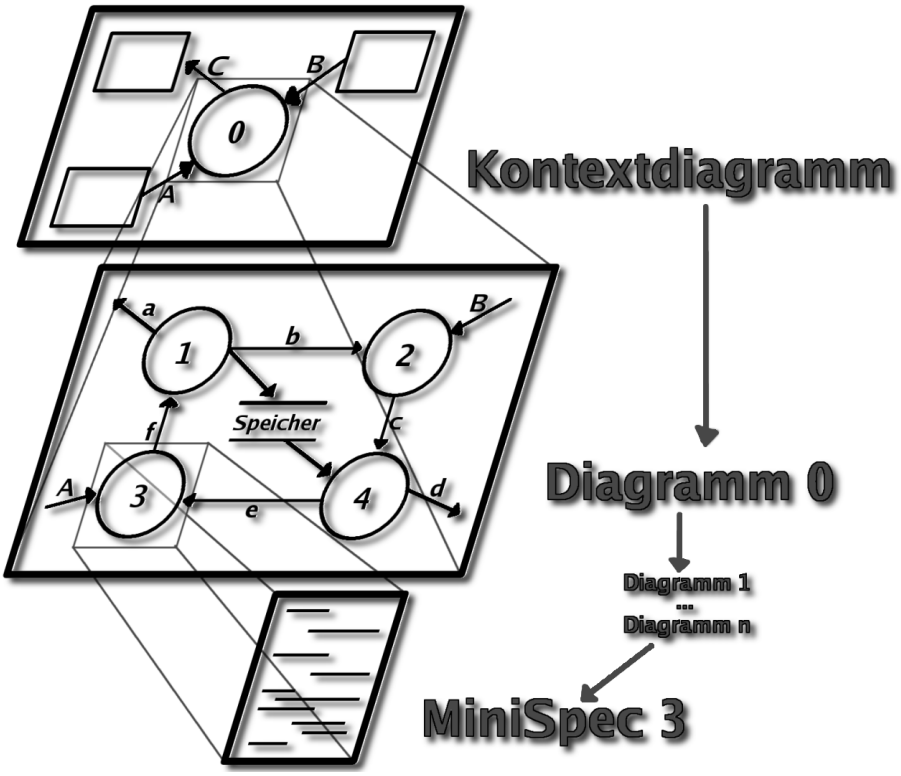
Damit ein Datenflussdiagramm für alle partizipierenden Parteien leicht verständlich bleibt, muss es selbst ohne eine Erfassung der textuellen Beschreibung die Grundzüge des Systemes wiedergeben können.

[4]CASE steht für Computer Aided Software Engineering. CASE-Werkzeuge unterstützen den Software-Ingenieur bei der Planung, Entwurf und Inplementierung sowie Dokumentation.

2.1.2 Das Hierarchiekonzept

Das System wird Schrittweise verfeinert. Die Verfeinerung wird solange weiter durchgeführt, bis sich die Prozesse nicht weiter verfeinern lassen, das heisst, sie primitiv sind. Primitive Prozesse werden sodann durch eine MiniSpec beschrieben.

Abbildung 3: Das Hierarchiekonzept



Untergeordnete Diagramme werden als Kinder (children), übergeordnete Diagramme als Eltern (parents) bezeichnet. Jedes Datenflussdiagramm wird durch hierarchisch gebildete Prozessnummern ausgezeichnet.

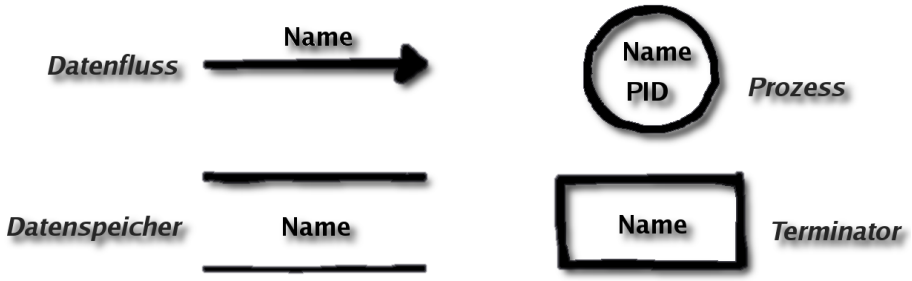
2.1.3 Begriffserklärung

Datenflussdiagramme beschreiben die Umwandlung von Eingabedaten in Ausgabedaten.

Es werden die folgenden Notationen verwendet:

Datenflüsse werden durch einen Pfeil mit der zugehörigen Beschreibung dargestellt. Die Richtung, in welche der Pfeil zeigt, gibt an, in welche Richtung die Daten fliessen.

Abbildung 4: Notationen für Datenflussdiagramme



Datenspeicher werden durch zwei parallele Striche dargestellt. Speicher bilden eine Ablagemöglichkeit für Daten, bei denen sich der Erstellungszeitpunkt vom Gebrauchszeitpunkt unterscheidet. Die Richtung des Datenflusses auf einem Datenspeicher gibt an, ob aus dem Speicher gelesen oder auf den Speicher geschrieben wird.

Terminatoren stehen für die Beziehungen des Systemes zur Aussenwelt. Sie können lediglich Daten aufnehmen und weiterleiten, diese aber nicht bearbeiten. Terminatoren dürfen nur im Kontextdiagramm Verwendung finden.

Prozesse werden durch Kreise, auch "Bubbles" genannt, dargestellt. Sie werden durch einen Namen und eine Nummer beschrieben und haben die Aufgabe, Eingabedaten in Ausgabedaten umzuwandeln. Prozesse arbeiten nur dann, wenn Ihnen die benötigten Datenflüsse zur Verfügung stehen. Ist ein Prozess nicht mehr weiter aufteilbar oder sinnvoll einfacher zu beschreiben, so wird er durch eine MiniSpec beschrieben.

2.1.4 Syntax und Semantik

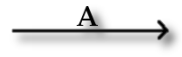
Syntax Ein Datenflussdiagramm sollte nicht mehr als sieben Prozesse enthalten. Jeder Datenfluss muss mit mindestens einem Prozess verbunden sein. Jeder Prozess und jeder Datenfluss, der nicht mit einem Datenspeicher verbunden ist, muss einen Namen tragen. Ein Prozess ohne Ausgang kann nicht exisitieren. Desweiteren kann es keine Datenflüsse zwischen den Terminatoren geben.

Semantik In Datenflussdiagrammen sollen Prozesse, Datenspeicher und Datenflüsse mit möglichst kurzen, prägnanten Namen versehen werden. Prozessnamen sollen aus einem Verb, das eine Aktion beschreibt und einem Substantiv gebildet werden. Datenflüsse sollen durch ein Substantiv beschrieben werden, welches keine Verarbeitung beschreiben darf. Der Speichername sollte auf den Inhalt des Speichers hinweisen.

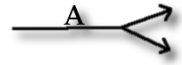
2.1.5 Datenflüsse

Ein Datenfluss ist vergleichbar mit einer Pipeline, durch die Daten transportiert werden. Jeder Datenfluss muss eine Bezeichnung erhalten und im Datenlexikon vermerkt werden.

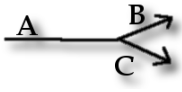
Die folgenden Datenflussmöglichkeiten sind möglich:



A fließt von links nach rechts



A wird in beide Zweige weitergeleitet



A teilt sich in *B* und *C*

A entsteht aus *B* und C

A fließt in beide Richtungen

2.1.6 Das Kontextdiagramm

Das Kontextdiagramm beschreibt die höchste Ebene in der Datenflussdiagrammhirarchie und den Anwendungsbereich des modellierten Systemes. Hier werden die Schnittstellen des Systemes zur Aussenwelt dargestellt. Das zu betrachtende System wird durch einen einzigen Prozess veralgemeinert, den Gesamtsystems-Prozess, welcher sodann die Nummer Null erhält. Der Gesamtsystems-Prozess ist die Zusammenfassung des nullten Diagrammes. Weiterhin kann das Kontextdiagramm keinen Speicher besitzen.

2.2 Datenlexikon

Für jeden Datenfluss und Speicher muss im Datenlexikon (Data Dictionary) eine Beschreibung in einer speziellen Notation, der Backus-Naur-Form erfolgen. Es ist lediglich die inhaltliche Bedeutung von Relevanz,

physikalische Informationen über den Speicher werden nicht berücksichtigt.

Abbildung 5: Backus-Naur-Form

= ist äquivalent

+ Sequenz; und

[..|..] Auswahl; entweder, oder

{} Wiederholung; m { } n, von m bis n-mal
{ ... } 5; höchstens fünf
2 { ... } 8; zwei bis acht

() Option

.. Kommentar

Wiederholt vorkommende Datenelemente werden über Hilfsdefintionen beschrieben und können dann wie Konstanten Wiederverwendung finden. Das Datenlexikon erlaubt die Überprüfung des Datenmodelles auf Rendundanz und Widerspruchsfreiheit.

2.3 MiniSpecs

Ein Prozess der nicht mehr weiter aufteilbar oder sinnvoll einfacher zu beschreiben ist, wird durch eine MiniSpec beschrieben. Hierbei kann jedes Mittel Berücksichtigung finden, welches einfach zu verstehen ist und den Prozess eindeutig beschreibt. Häufig wird in diesem Fall auf einen Pseudocode[5] zurückgegriffen.

Abbildung 6: Beispiel für eine MiniSpec

```
* Unbenutzte User-Accounts löschen. *

Alle User-Accounts aus Datenbank selektieren;
if letzter Login > 365 Tage
     then verschicke Benachrichtigungsmail
else if letzter Login > 400 Tage
     then lösche Account
```

[5]Ein Pseudocode ist eine einfach verständliche, an eine Programmiersprache angelehnte Sprache.

Prozesse in den Datenflussdiagrammen sollten soweit verfeinert werden, dass die dazugehörige MiniSpec auf einer Seite niedergeschrieben werden kann. MiniSpecs beschreiben grundsätzlich, wie die Eingabedaten der Prozesse auf der untersten Ebene der Datenflussdiagramme in Ausgabedaten transformiert werden. Eingabedaten sowie Ausgabedaten sind Bestandteil der MiniSpec.

3 Ausklang

3.1 Stärken

Tom DeMarco selbst beschrieb die Stärken der Strukturierten Analyse durch folgende Kernaussagen:

Graphisch Ein aussagekräftiges Bild ist einfach zu verstehen.

Partitionierung Kann in ein *Top-Down*-Ansatz zerlegt werden.

Striktheit Strikte Dokumentation und Spezifikation.

Veränderbar Leicht veränderbar, Rendundanzen werden vermieden.

Iterativ Teile der Spezifikationen können separat addressiert werden.

Logisch Sichert die Anforderungen vor Veränderungen in der physikalischen Architektur.

Verständlich Leicht nachvollziehbar; für Leser aus allen Fachbereichen verständlich.

Die Stärken der Strukturierten Analyse liegen also in der Kombination bewährter Konzepte. Die hierarchische Gliederung sorgt für eine gute Übersichtlichkeit, die graphische Notation ist leicht erlernbar und die Datenflussdiagramme sind bis zu einer gewissen Größe leicht nachvollziehbar.

3.2 Schwächen

Aus heutiger Sicht hat die Strukturierte Analyse viele Schwächen, weshalb sie auch seltener in neuen Projekten Verwendung findet. Bei hochgradig komplexen Systemen mit vielen Schnittstellen ist es sehr kompliziert, dass Kontextdiagramm der ersten Ebene zu erstellen. Auch die direkte Zerlegung des Kontextdiagramms im Sinne des *Top-Down*-Vorgehens funktioniert nicht immer. Die simple grafische Notation legt dem Betrachter viele Interpretationsmöglichkeiten offen. Auch die Benennung der grafischen Elemente durch treffende Aussagen ist für das Verständnis der Diagramme von (zu) großer Bedeutung. Weiterhin fehlt

ein Konzept für die Analyse der gespeicherten Daten und das Zeitverhalten wird ebenso nicht berücksichtigt.

Literatur

[1] Broy, Denert *"Software Pioneers"*. Springer Verlag, 2002

[2] DeMarco, Tom *"Structured Analysis and System Specification"*. Prentice-Hall - Yourdon, 1978

[3] DeMarco, Tom *"Peopleware, 2nd Edition"*. Dorset House, 1999

[4] Wirsing, Martin *"Methoden des Software Engeneering"*. http://www.pst.informatik.uni-muenchen.de/lehre/WS0304/mse/REGrundlagen6p.pdf

[5] Prof. Dr. Ing. habil. Dipl.-Math. Klaus Fähnrich, *"Vorlesung Softwaretechnik"*. http://ais.informatik.uni-leipzig.de/download/2002w_v_swt/2002w_swt_v_08-4.pdf