Dipl.-Kfm.
Friedrich Kiltz

Fachliche Analysemethoden/Design (strukturiert)

Celecta GmbH

für die Ausbildungskooperation

Inhalt

1 Seminarübersicht

Die Analyse für ein Projekt ist die Vorstufe der Realisierung. Eine reine Analyse ohne deren Realisierung wäre vergleichbar mit einem ewigen Vorspiel bei dem es nie zu Koitus kommt.

Um solch einen schalen Geschmack zu vermeiden werden wir, nachdem wir die einzelnen Methoden in der Theorie kennen gelernt haben, ein existierendes Projekt analysieren und ich werde als Advocati Diaboli die Auswirkungen unserer Ungenauigkeiten fest legen. Zur Unterstützung werden wir uns einige Entscheidung in der Realisierung mit PHP ansehen.

Somit unterteilen sich die folgenden Abschnitte immer in

- Darstellung der zu Grunde liegenden Theorie
- Vertiefung durch ein eingängiges Beispiel
- Nutzung des Verfahrens an einem existierenden Projekt
- Besprechung der Ergebnisse und deren Auswirkungen an unserem Projekt

Zu Anfang des Kurses werden wir uns ein Projekt aussuchen, das wir dann durch alle Stufen begleiten werden. Außerdem werden wir die Grundlagen von PHP betrachten. Informationen zu PHP können vielfältig im Netz bezogen werden.

2 Funktionenmodell

2.1 Beschreibungen / Inhalt

Funktionenmodelle beschreiben und strukturieren die Funktionalität eines Systems. Diese Modelle geben Auskunft darüber, **was** ein System kann.

Es können folgende Teilbereiche modelliert werden:

- Der **Steuerfluss** in einem System oder in einer Funktion eines Systems. Dies können Ablauf- oder Aufrufstrukturen sein.
- Der **Datenfluss** in einer Funktion oder zwischen Funktionen des Systems
- Der **Arbeitsfluss** in einem Arbeitsprozess.

2.2 Syntax

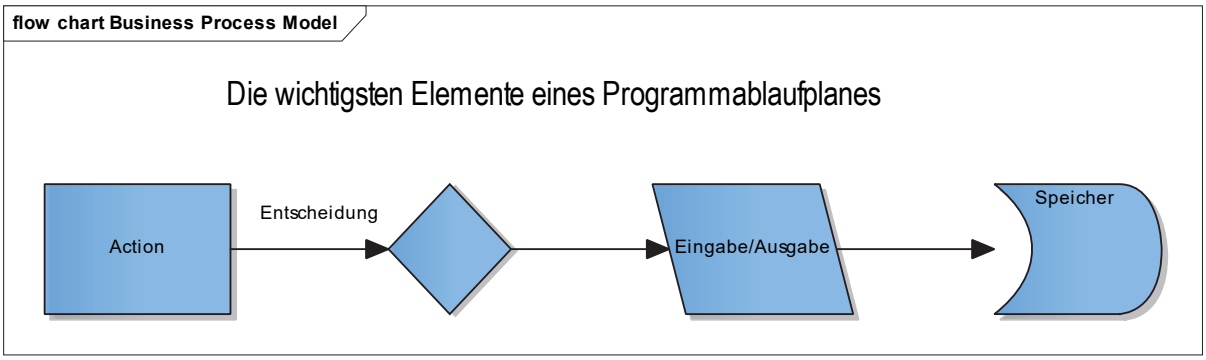
Funktionenmodelle werden je nach dem darzustellenden Teilbereich als

- Programmablaufplan
- Strukturierte Modelle von Programmabläufen wie z.B.
  - Nassi-Schneidermann-Diagramme
  - Pseudocode
- Prozeduraufrufgraphen
- Entscheidungstabellen
- UML-Aktivitätsdiagramme

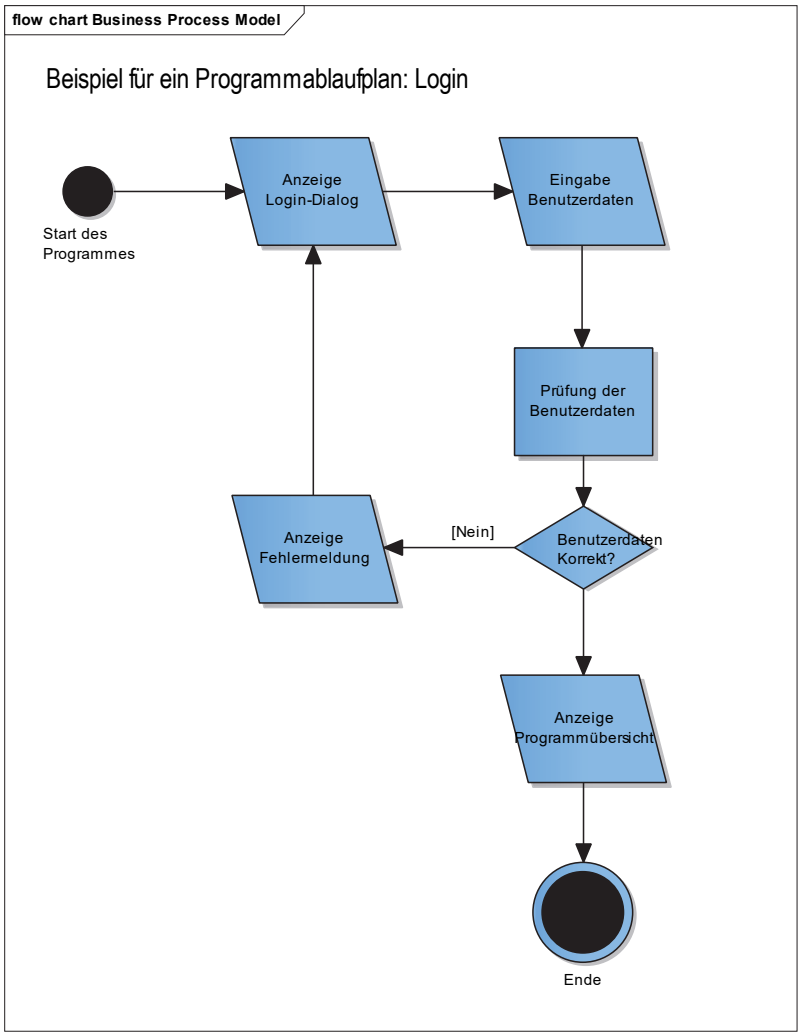
dargestellt werden. Im Prinzip wird jedes sequentielle Programm (und ein solches liegt bei strukturierter Programmierung vor) mit den Elementen *Anweisung*, *Alternative* und *Iteration* beschrieben werden.

Programmablaufplan

Programmablaufpläne (auch Flow Chart genannt) sind eines der ältesten (und eigentlich auch ein veraltetes) Mittel zur Visualisierung eines Funktionsmodelles. Eines seiner Probleme liegt darin, dass ein Programmablaufplan gerne zu einem Spagetti-Diagramm wird. Die grundsätzlichen Elemente eines Programmablaufplanes sind:



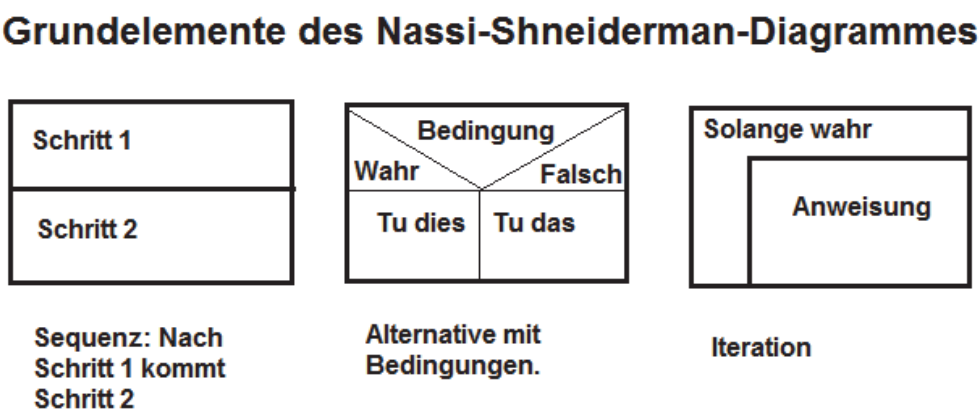
Die Elemente werden mit Pfeilen verbunden, die den Steuerfluss anzeigen. Teilweise wird auch zwischen Eingabe und Ausgabe unterschieden. Dabei wird für die Ausgabe ein Symbol genutzt, das mit einem durchgerissenen DIN-A-4-Blatt vergleichbar ist.



Weitere Informationen unter http://de.wikipedia.org/wiki/Programmablaufplan

Nassi-Shneiderman-Diagramm

Das Nassi-Shneiderman-Diagramm zeigt den Programmablauf in Strukturblöcken auf. Hierbei werden Anweisungen mit einem Rechteck dargestellt und Verzweigungen führen zu einer Teilung des Blockes. Der Block soll solange geteilt werden bis das Gesamtproblem in kleine lösbare Probleme zerlegt ist.



Weitere Informationen unter http://de.wikipedia.org/wiki/Nassi-Shneiderman-Diagramm

Pseudocode

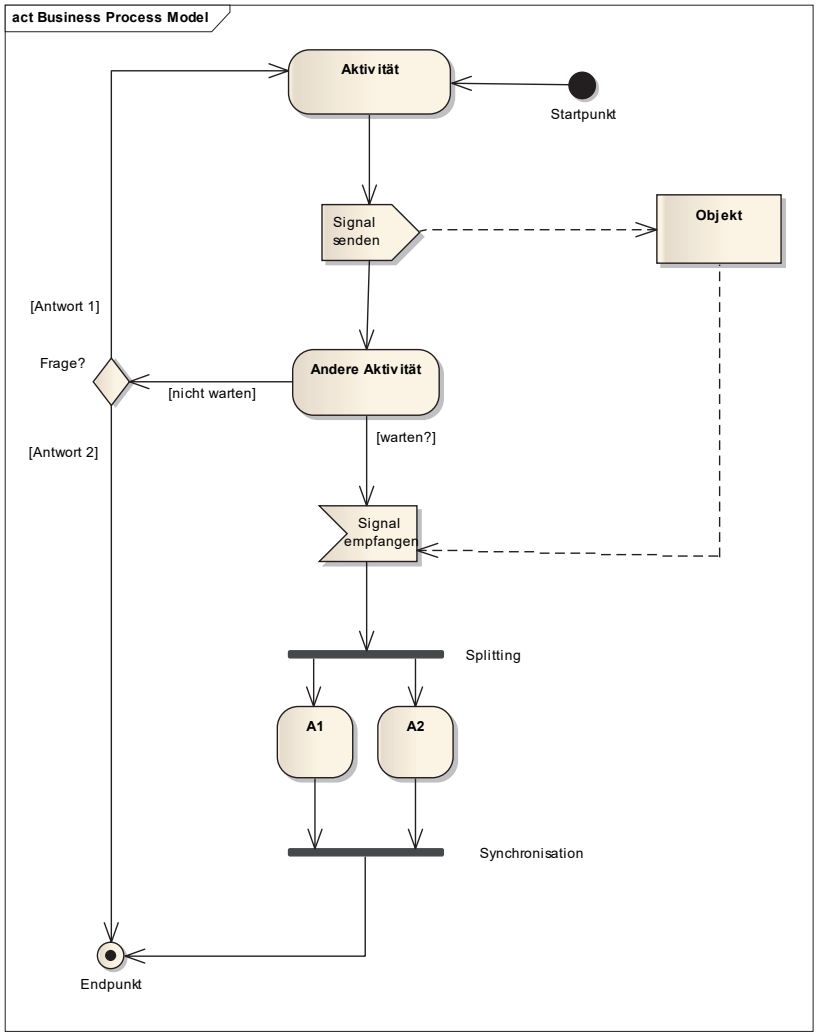
Pseudocode ist die Beschreibung von Anweisungen, Bedingungen und Iteration mit einer stark reduzierten Sprache. Ein Beispiel dazu könnte folgendermaßen aussehen:

```
Zeige Dialog
Gebe Benutzerdaten ein
Prüfe Benutzerdaten
Wenn Benutzerdaten falsch
     Fehlermeldung anzeigen
     Loginformular anzeigen
Sonst
     Startseite anzeigen
Ende Wenn
```

Weitere Informationen unter http://de.wikipedia.org/wiki/Pseudocode

UML-Aktivitätsdiagramme

UML-Aktivitätsdiagramme sind aktuell die am weitesten verbreite Modellart für die Darstellung von funktionellen Abläufen. Sie gehörten zu der Gruppe der Verhaltensdiagramme in der UML. Exemplarisch sollen in dem nachfolgenden Diagramm die wichtigsten Elemente sinnfrei angezeigt werden:



Für weiter Details s. https://de.wikipedia.org/wiki/Aktivitätsdiagramm

2.3 Nutzung

Bei der Erstellung strukturierter Ablaufmodelle wird nach dem klassischen Ansatz eine schrittweise Verfeinerung verfolgt. Dabei durchläuft man folgende Schritte:

1. Wähle eine Aktion, welche den gesamten Ablauf repräsentiert
2. Zerlege diese Aktion entsprechend der Problemstellung in eine Sequenz, eine Alternative oder eine Iteration.
3. Wiederhole Schritt 2 für jede beim Zerlegen neu entstandene Aktion, bis das Modell hinreichend detailliert ist.

Bei dieser Vorgehendweise können sehr leicht Zerlegungsfehler auftreten, die später nicht mehr reparierbar sind.

Eine bessere Alternative ist:

- Teilprobleme identifizieren
- Alle Teilprobleme durch schrittweise Verfeinerung modellieren
- Teilmodelle durch Verschachtelung zusammensetzen.

3 Datenmodell

3.1 Entitäten, Attribute, Beziehungen

In der Analyse-Phase werden häufig Modelle zu den benötigten Daten erstellt. Diese Modelle sollen die Kommunikation zwischen Entwickler und Fachbereich/Anwender verbessern.

Zur Ermittlung und Analyse der Datenmodelle bedient man sich recht häufig der Entity-Relationship-Modelle (http://de.wikipedia.org/wiki/Entity-Relationship-Modell). Dabei ist eine Entity (Entität) ein Objekt oder eine Gegebenheit der realen Welt (Kunde, Rechnung, Buchung etc.). Diese Entitäten können in bestimmten Beziehungen zu anderen Entitäten stehen (ein Kunde **kauft** eine Ware). Hierbei kann auch die Kardinalität/Multiplizität mit angegeben werden. Weiterhin werden die Entitäten durch Attribute beschrieben. Der Typ der Attribute ist dabei im Normalfall nicht von Bedeutung.

In der erweiterten Version des Entity-Relationship-Modelles gibt es auch die Möglichkeit Generalisierungen (Vererbung, ist-ein) und Aggregationen (hat-ein) dar zu stellen.

Zur graphischen Darstellung von Entity-Relationship-Modellen bedient man sich normalerweise der Chen-Notation (http://de.wikipedia.org/wiki/Chen-Notation) . Diese ist in Wikipedia so gut beschrieben, dass sie hier keiner weiteren Ausführung bedarf.

3.2 Normalisierung

Zur Verbesserung des Datenbank-Designs gibt es die Regeln der Normalisierung. Der zentrale Punkt der Normalisierung ist die Vermeidung von Redundanzen, also von unnötigen Wiederholungen von Daten in einer Datenbank. Durch die Vermeidung von Redundanzen sinkt das Risiko, dass bei Änderungen nur ein Teil der betroffenen Daten geändert werden und ein anderer zu ändernde Teil bei der Aktualisierung vergessen wird. Diese Situation führt zu Anomalien in der Datenbank.

Dies möge an einem kleinen Beispiel veranschaulicht werden:

Angenommen wir speichern Rechnungen ab. Bei einem schlechten Design würden wir in der Tabelle „Rechnungen“ die Namen des Kunden und seine Positionen abspeichern.

Name	Straße	Plz/Ort	Nr	Menge	Artikel	Einzelpreis
Friedrich Kiltz	Schwarzed 1	84549 Engelsberg	123	1	Hose	75,20
Friedrich Kiltz	Schwarzed 1	84549 Engelsberg	123	2	Hemd	30,00
Friedrich Kiltz	Schwarzed 1	84549 Engelsberg	127	1	Socken	12,30

Wie man oben leicht erkennen kann, wiederholt sich der Name und die Adresse des Kunden für jede Rechnungsposition. Dies hat zwei große Nachteile:

1. Man verschwendet durch die Wiederholung Speicherplatz.
2. Bei einer Adresskorrektur müssten alle Datensätze geändert werden.

Die Normalisierung unterscheidet 5 Normalformen, die man als Regeln für den Aufbau der Datenbanken nutzen kann. Die wichtigen Normalformen dabei sind die ersten drei, die ich hier behandeln möchte.

1. Normalform

Die erste Normalform besagt:

Jedes Attribut der Relation muss einen atomaren Wertebereich haben.

Obiges Beispiel verstößt gegen diese Regel. Name muss nach der ersten Normalform in Vorname und Nachname, PLZ/Ort in die Felder PLZ und Ort aufgeteilt werden. Bezüglich der Straße und Hausnummer könnte man sich streiten.

2. Normalform

Die zweite Normalform lautet:

Eine Relation ist in der zweiten Normalform, wenn die erste Normalform vorliegt und kein Nichtschlüsselattribut funktional abhängig von einer echten Teilmenge eines Schlüsselkandidaten ist.

D.h. Sofern sich der Schlüssel aus mehreren Teilen zusammen setzt dürfen in den restlichen Attributen der Tabelle nur die Werte stehen, die sich alleine auf den zusammengesetzten Schlüssel beziehen. Würde im obigen Beispiel noch eine Positionsnummer zur Rechnungsnummer gehören und die Rechnungsnummer +Positionsnummer der Schlüssel darstellen, dürften die Angaben zur gesamten Rechnung (Kunde, Datum etc. nicht in dieser Tabelle enthalten sein. Siehe dazu auch das CD-Beispiel bei Wikipedia.

3. Normalform

Die dritte Normalform lautet:

Die dritte Normalform ist genau dann erreicht, wenn sich das Relationenschema in 2NF befindet, und kein Nichtschlüsselattribut von einem Schlüsselkandidaten transitiv abhängt.

Dies bedeutet, dass aus keinem Nichtschlüsselattribut ein anderes Nichtschlüsselattribut folgen darf. Ein gutes Beispiel hierfür finden wir wieder auf Wikipedia mit dem CD-Beispiel. Hier ist das Gründungsjahr vom Interpreten abhängig und der Interpret ein Nichtschlüsselattribut. Somit sollte Interpret mit „seinem“ Gründungsjahr in eine eigene Tabelle ausgelagert werden und per Fremdschlüssel referenziert werden.

3.3 Auflösungen m:n

Eine m:n Relation liegt vor, wenn zwei unterschiedliche Entity-Typen zu beiden Seiten mehrfach zugewiesen werden können. Hierzu ein kleines Beispiel:

Angenommen wir haben Benutzer eines Systems, die unterschiedliche Rollen haben können.

Benutzer
Willi
Helmut
Gerhard

Rollen
Benutzer
Administrator
Tester

Wenn nun der Benutzer *Helmut* sowohl *Administrator* als auch *Tester* und der *Willi Benutzer* und *Tester* ist und *Gerhard* nur *Benutzer* ist könnten wir das folgendermaßen darstellen:

Benutzer	Rollen
Willi	Benutzer, Tester
Helmut	Administrator, Tester
Gerhard	Benutzer

Gegen welche Regeln der Normalisierung verstößt dieser Ansatz?

Die Lösung für das Problem liegt in einer sogenannten JoinTable. Erst geben wir unseren Entites eine ID:

ID	Benutzer
1	Willi
2	Helmut
3	Gerhard

ID	Rollen
1	Benutzer
2	Administrator
3	Tester

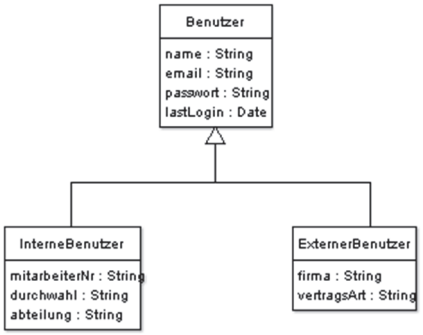
Nun werden die Relationen zu den beiden in einer eigenen Tabelle – einer JoinTable – abgebildet:

ID_Benutzer	ID_Rolle
1	1
1	3
2	2
2	3
3	1

Somit wird für jede Relation ein Datensatz in der JoinTable eingefügt.

3.4 Generalisierung / Spezialisierung

Auch bei Daten einer Datenbank kann Vererbung genutzt werden. Nehmen wir an, wir haben eine Entity „Benutzer“, die entweder ein „Interner Benutzer“ oder ein „Externer Benutzer“ ist.



Wie kann man nun diese Vererbung in einer Datenbank-Struktur abbilden und welche Auswirkungen haben die unterschiedlichen Möglichkeiten?

In der Datenbank kann nun diese Konstellation in folgenden Arten abgespeichert werden:

- Eine Tabelle für die ganze Klassenhierarchie
- Eine Tabelle pro Klasse

- Eine Tabelle pro konkreter Klasse (wenn Benutzer abstrakt sei)

Eine Tabelle für die ganze Klassenhierarchie

Die einfachste Variante ist, für alle Attribute, die vorkommen können in einer Tabelle entsprechende Spalten einzurichten. Um zu unterscheiden, von welchem Typ das Ergebnis ist, wird eine eigene Spalte für den Typ (Diskriminatorspalte) eingeführt. Die optionalen Properties müssen nullable sein.

Der Nachteil dieser Strategie ist es, dass bestimmte Felder nullable gemacht werden müssen, die für die konkrete Entity gar nicht nullable sind. Der Vorteil ist, dass diese Strategie einfach ist und die beste Performance mitbringt.

Eine Tabelle pro Klasse

Die Strategie „Joined Subclasses“ stellt in der Datenbank für jede Klasse eine eigene Tabelle zur Verfügung, die mit 1:1-Relationen verbunden werden. Dabei werden die Daten durch gleich Id's synchronisiert.

Diese Strategie hat eine schlechtere Performance, da bei jeder Abfrage mehrere Joins notwendig sind. Manuelle Nutzung dieser Tabellenstruktur sind nicht ganz einfach. Der Vorteil dieser Strategie ist die natürlich Umsetzung des OO-Modelles.

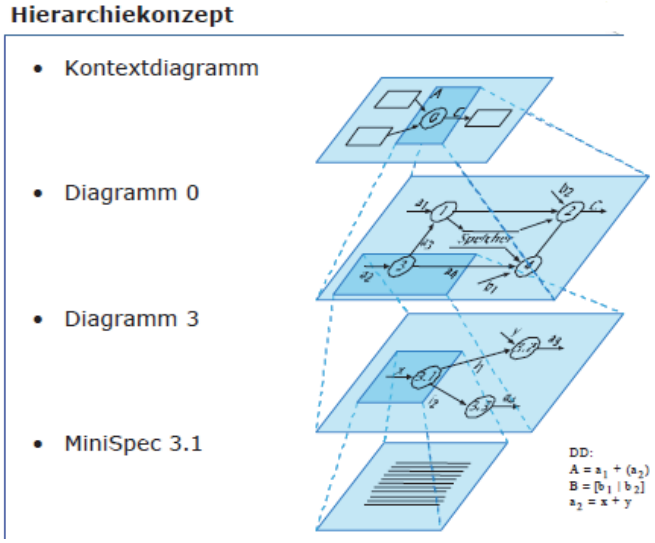
Eine Tabelle pro konkreter Klasse

In der Strategie „Tables per Class“ wird für jede konkrete Entity eine Tabelle angelegt, die die Felder der Superklasse mit beinhaltet. Von diesem Ansatz ist abzuraten, da man doppelte Spalten hat und keine eindeutigen Id's für die Superklasse hat. Bei polymorphen Abfragen muss ein SQL-UNION benutzt werden, was die Performance sehr beeinträchtigt.

4 Kontextdiagramm

4.1 Beschreibungen / Inhalt

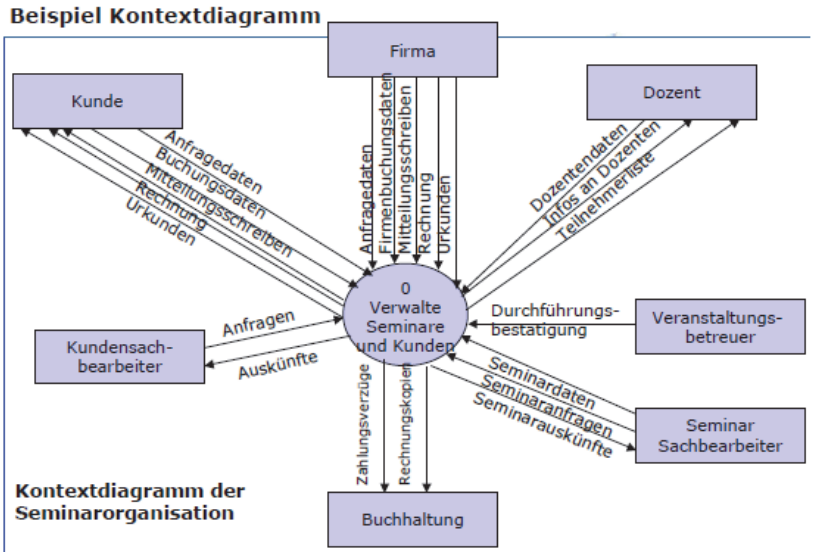
Mit einem Kontextdiagramm wird die Modellierung einer Systemumgebung in einer frühen Entwurfsphase wieder gegeben. Ein Kontextdiagramm ist ein abstraktes Datenflussdiagramm, das die Schnittstellen unseres Systems zu seiner Umwelt abbildet. Jeder Prozess wird in untergeordnete Diagramme verfeinert. Sofern keine weitere Verfeinerung möglich ist wird eine textuelle Mini-Spezifikation erstellt.



(aus Uni Leipzig, LV Softwaretechnik, WS 02/03)

4.2 Syntax, Darstellung

Ein Kontextdiagramm enthält genau einen Prozess, der als Kreis in der Mitte des Diagrammes dargestellt wird. Die Komponenten, die mit dem Prozess interagieren werden als Rechtecke dargestellt. Zwischen den Komponenten und dem Prozess bestehen Datenflüsse, die mit einem Pfeil dargestellt werden.



(aus Uni Leipzig, LV Softwaretechnik, WS 02/03)

Bei der UML wird das Kontextdiagramm durch ein Anwendungsfalldiagramm ersetzt

5 Geschäftsprozessmodellierung

5.1 Beschreibung

Unternehmenssoftware unterstützt die Geschäftsprozesse des Unternehmens. Von daher ist es wichtig die Geschäftsprozesse des Unternehmens zu identifizieren und mit ihren Beziehungen graphisch dar zustellen (zu modellieren).

Auslöser für Geschäftsprozesse sind im Normalfall **Geschäftspartner**. Somit sollte mit der Identifizierung der Geschäftspartner begonnen werden. Geschäftspartner sind Akteure die außerhalb des Unternehmens stehen und aktiv oder passiv Geschäftsprozesse initiieren. Typische Geschäftspartner sind Kunden, Lieferanten und Berater sowie die Behörden.

Die eigentlichen Ereignisse, die von den Geschäftspartnern ausgelöst werden sind Geschäfts-anwendungsfälle. Zusammengehörende Geschäftsanwendungsfälle können zu Geschäftsprozessen zusammengefasst werden.

5.2 Darstellung

Die Art der Darstellung der Ergebnisse einer Geschäftsprozessmodellierung sind nicht genormt. Es gibt unterschiedliche Ansätze für die Darstellung. Vergleiche hierzu auch http://de.wikipedia.org/wiki/Geschäftsprozessmodellierung.

Ein interessanter Ansatz der Modellierung finden wir in dem Buch „Objektorientierte Geschäftsprozessmodellierung mit der UML“ von Mitarbeitern der Firma OOSE. Dieser Ansatz nutz Anwendungsfall- und Aktivitätsdiagramme für die Modellierung.

Ein weiterer Ansatz ist die Business Process Model and Notation (BPMN), die unter http://de.wikipedia.org/wiki/BPMN recht gut beschrieben ist.

Die grafischen Elemente der BPMN werden eingeteilt in

- Flow Objects - die Knoten (Activity, Gateway und Event) in den Geschäftsprozessdiagrammen
- Connecting Objects - die verbindenden Kanten in den Geschäftsprozessdiagrammen
- Pools und Swimlanes - die Bereiche, mit denen Aktoren und Systeme dargestellt werden
- Artifacts - weitere Elemente wie Data Objects, Groups und Annotations zur weiteren Dokumentation

6 Vorgehensmodelle

6.1 Prozessmodelle

Prozessmodell beschreiben den allgemeinen Entwicklungsplan zur Realisierung einer Software. Das Prozessmodell legt die Reihenfolge der Arbeitsschritte fest und bestimmt die Verantwortlichkeiten und die Kompetenzen der Mitarbeiter.

Ziel eines Prozessmodelles ist der Versuch die Komplexität der Softwareerstellung zu beherrschen. Dazu wird der Entwicklungsprozess in Phasen unterteilt, die wie im **Wasserfallmodell** (s. z.B. https://de.wikipedia.org/wiki/Wasserfallmodell) in einer bestimmten Reihenfolge durchlaufen werden oder wie in dem **Spiralmodell** (s. z.B. https://de.wikipedia.org/wiki/Spiralmodell) iterativ-verfeinernd durchlaufen werden.

Ein grundsätzliches Problem der Prozessmodelle ist, das zu Beginn des Projektes die genau benötigten Funktionen meist nicht genau bekannt sind oder sich im Laufe der Entwicklung verändern. Somit überholt sich die vollständige Analyse des Projektes vor dem eigentlichen Entwicklungsbeginn darin, dass zur Zeit der Entwicklung der Bausteine die Anforderungen der Analyse sich wieder geändert haben.

Alternativen zu den Prozessmodellen sind Entwicklungsphilosophien wie Extreme Programming oder die Elemente der agilen Softwareentwicklung wie Scrum oder Kanban.

6.2 Agile Methoden

Um das Problem der vollständigen vorherigen und starren Analyse zu umgehen hat sich die agile Softwareentwicklung gebildet. Der Kernpunkt der agilen Softwareentwicklung zeichnet sich durch wenig Bürokratie aus. Der Entwickler wird als selbständig handelnder und verantwortungsbewusst agierender Teil des Entwicklungsprozesses angesehen (und nicht durch die Bürokratie „gegängelt“).

Das Fundament der agilen Softwareentwicklung liegt in ihrem Manifest:

1. Individuen und Interaktionen gelten mehr als Prozesse und Tools.
2. Funktionierende Programme gelten mehr als eine ausführliche Dokumentation.
3. Die stetige Zusammenarbeit mit dem Kunden steht über Verträgen.
4. Der Mut und die Offenheit für Veränderungen steht über dem Befolgen eines festgelegten Plans.

Beispiele für agile Prozesse sind **Scrum** und **Extreme Programming** (XP).

Scrum

Scrum ist so einfach, das es auf einem Bierdeckel erklärt werden kann.

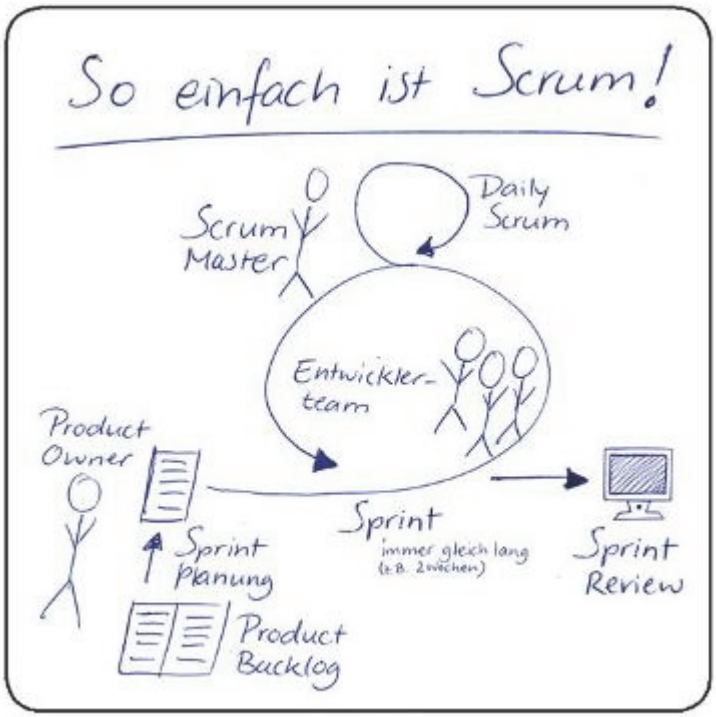


Bild 1: Die Grundlagen von Scrum sind so einfach, dass sie auf einen Bierdeckel passen.

(aus http://inticon.de/cms/seminare/scrum.php)

Es gibt ein paar Rollen:

- **Product Owner**
  Kennt die Produktvision, repräsentiert den Kunden/Nutzer der Software
- **Entwicklungsteam**
  Die Entwickler des Projektes
- **ScrumMaster**
  Gehört nicht zum Entwicklungsteam, hält diesem aber den Rücken frei.

Und ein paar Artefakte:

- **Product Backlog**
  Die priorisierte Liste der Anforderungen.
- **Sprint Backlog**
  Die Planung für einen Sprint

Ein paar Meetings:

- Sprint Planning:
  Was soll im nächsten Sprint gemacht werden?
- Daily Scrum:
  15 Min-Meeting: Was tat ich, was will ich tun und was behindert mich?

- Sprint Review:
  Am Ende eines Sprints: Waren wir gut?

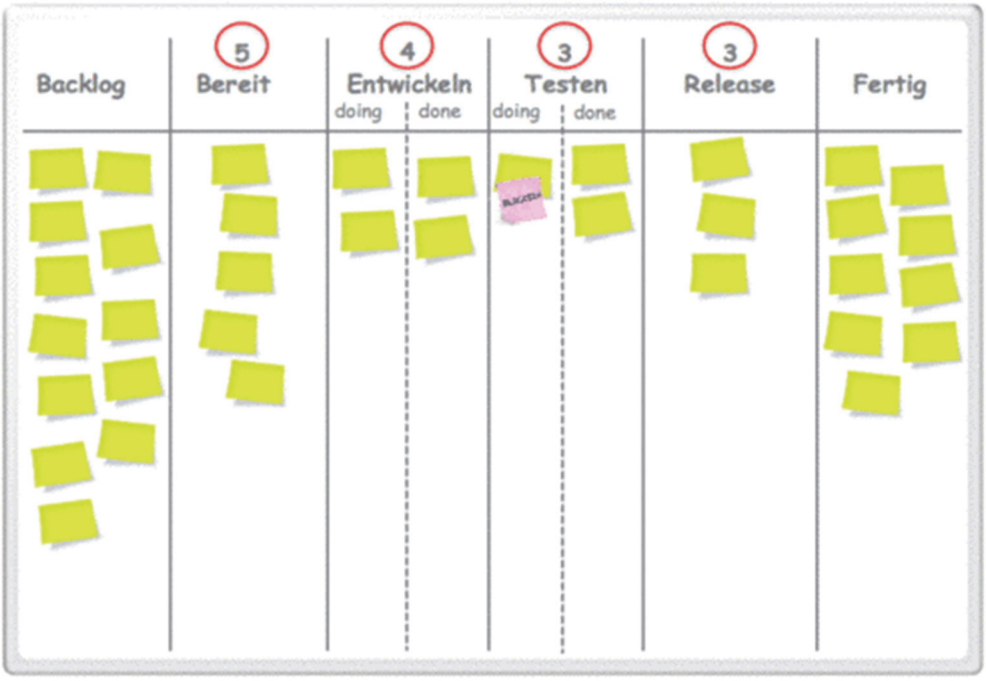
Und natürlich den Sprint. Der Sprint ist ein gleichbleibender Entwicklungszyklus zwischen einer und 4 Wochen.

Ganz so einfach ist es nicht. In der Theorie gibt es noch ein paar mehr Rollen, ein paar mehr Meetings und noch ein paar Artefakte. Siehe dazu z.B. http://de.wikipedia.org/wiki/Scrum. Das schwierigste bei Scrum ist die Einführung bei einem Unternehmen, die Selbständigkeit und Verantwortlichkeit der Softwareentwickler, die vermeintlich schlechtere Planbarkeit durch den Kunden, die Bereitschaft des Auftraggebers Kontrolle abzugeben.

Kanban

Kanban ist eine Methode den Workflow zu visualisieren. Hierbei wird auf einer Tafel drei oder mehr Spalten angelegt. Minimum sind „To-Do“, „Doing“ und „Done“.

In den einzelnen Spalten werden die Tasks von links nach rechts verschoben. Dabei ist es sinnvoll für bestimmte Spalten ein Maximum einzuführen, damit nicht zu viele Aufgaben unerledigt „in Bearbeitung“ sind. Eine etwas umfangreichere Unterteilung könnte auch folgendermaßen ausschauen:



(aus https://www.it-agile.de/wissen/einstieg-und-ueberblick/kanban/)

Die Zahlen in den roten Kreisen sind das Maximum an Tickets für die entsprechenden Spalten.