Software-Definition

Softwaretechnik

Andreas Zeller
Lehrstuhl für Softwaretechnik
Universität des Saarlandes, Saarbrücken

2005-10-27

Produkt-Definition

Die Anforderungen an ein Software-Produkt sind von ihrer Natur her

vage, unzusammenhängend,
unvollständig und widersprüchlich

Ziel der *Software-Definition* ist es, aus diesen Anforderungen eine möglichst

- vollständige,
- konsistente und
- eindeutige

Produkt-Definition zu erstellen
(= Grundlage für das zu erstellende Produkt)

Produkt-Definition

Grundlage für die Definition sind die gesammelten Anforderungen – typischerweise in Form von *Use Cases*.

- Im *Extreme Programming* werden die Use Cases in *Stories* umgewandelt, die ihrerseits zu *Testfällen* werden.
- In der herkömmlichen Entwicklung wird aus den Use Cases zunächst ein *Pflichtenheft* erstellt, das die Anforderungen an die fertige Software festlegt.

Ein Pflichtenheft ist Vertragsgrundlage!

Das Pflichtenheft

Typischer Aufbau:[1]

1 Zielbestimmung

2 Produkt-Einsatz

3 Produkt-Umgebung

4 Produkt-Funktionen

5 Produkt-Daten

6 Produkt-Leistungen

7 Benutzungsoberfläche

8 Qualitäts-Zielbestimmung

9 Testszenarien

10 Entwicklungs-Umgebung

11 Ergänzungen

Das Pflichtenheft – Zielbestimmung

1 Zielbestimmung

Mußkriterien Welche Leistungen sind für das Produkt unabdingbar? *Beispiel Stundenplanverwaltung: Verwaltung von Lehrern, Fächern, Räumen, Klassen; Eingabe der Sollvorgaben, Ausgabe eines Stundenplanes; Ausdruck*

Wunschkriterien Welche Leistungen sind erstrebenswert? *Interaktive Optimierung, Ausgabe im HTML-Format*

Abgrenzungskriterien Welche Ziele sollen bewußt *nicht* erreicht werden? *GUI-Skins oder -Themes, Programmierbarkeit durch Endbenutzer, Schulen mit mehreren Schulformen (Gesamtschule)*

Das Pflichtenheft – Einsatz

2 Produkt-Einsatz

Anwendungsbereiche Zu welchem Zweck wird das Produkt eingesetzt?

Zielgruppen Von wem soll das Produkt eingesetzt werden? Welches Qualifikationsniveau des Benutzers wird vorausgesetzt? *Passant, Endanwender am Arbeitsplatz, System-Administrator, Programmierer*

Betriebsbedingungen Wie ist die physikalische Umgebung? Die Betriebszeit? Aufsicht? *Internet-Server, auf Boot, in Maschinenhalle, Desktop-Computer*

[1] nach Balzert, Lehrbuch der Software-Technik

Das Pflichtenheft – Umgebung

3 Produkt-Umgebung

Software Welche Software-Systeme sollen auf der Zielmaschine *zur Laufzeit* zur Verfügung stehen? *Linux 2.4/x86, libc6 (≥ 4.0), libglub1.2 (≥ 1.2.0), libglob1.2 (≥ 1.2.10-4)*

Hardware Welche Hardware-Komponenten werden benötigt? *8fach Opteron Dual-Core 2.0 GHz, Arbeitsspeicher 32GB, Plattenplatz 2TB oder mehr*

Orgware Welche organisatorischen Bedingungen müssen erfüllt sein? *Backup, statische IP-Adresse, Freischaltung von Ports in der Firewall.*

Das Pflichtenheft – Funktionen

4 Produkt-Funktionen

Funktion 1

Funktion 2 **usw.**

Was leistet das Produkt aus Benutzersicht?

Wichtig: Nicht das *Wie*, sondern das *Was* wird hier definiert.

Diese Funktionen können z.B. aus den Use-Cases entnommen werden

Jede Einzelanforderung muß gesondert gekennzeichnet sein (etwa als /F10/, /F20/, usw.)

Anforderungen sind möglichst zu *begründen*
(„Warum ist diese Funktion wichtig?“)

Das Pflichtenheft – Daten

5 Produkt-Daten Was speichert das Produkt (langfristig) aus Benutzersicht?

Auch hier sollten die zu speichernden Daten gesondert gekennzeichnet sein (etwa als /D10/, /D20/, usw.)

Hier werden häufig auch bereits Aussagen über das *Datenformat* getroffen, zum Beispiel durch Angabe einer XML Document Type Definition.

```
<!-- This is the DTD for font configuration files -->
<!ELEMENT alias (family*, prefer?, accept?, default?)>
<!ELEMENT prefer (family)*>
<!ELEMENT accept (family)*>
<!ELEMENT default (family)*>
<!ELEMENT family (#PCDATA)>
<!ATTLIST family xml:space (default|preserve) 'preserve'>
```

Das Pflichtenheft – Leistungen

6 Produkt-Leistungen

Welche zeit- und umfangsbezogenen Anforderungen gibt es?

Kennzeichnung als /L10/, /L20/, usw.

HTTP-Anfrage pro Minute, Anzahl Benutzer, Anzahl Artikel.

7 Benutzungsoberfläche

Was sind die grundlegenden Anforderungen an die Benutzeroberfläche?

Umfaßt Bildschirmlayout, Drucklayout, Tastaturbelegung, Dialogstruktur, usw.

8 Qualitäts-Zielbestimmung

Verweis auf gängige Normen und Standards.

Das Pflichtenheft – Tests

9 Testszenarien

Testfall 1
***Testfall 2* usw.**

Was sind typische Szenarien, die das Produkt erfüllen muß?

Ein Szenario sollte eine *Schritt für Schritt-Anleitung* für den Tester sein:

- /T10/ Wählen Sie das Objekt 1 aus /F10/.
- /T20/ Löschen Sie es /F25/ …

Die Testfälle sind gekennzeichnet durch /T10/, /T20/, usw.

Testszenarien sind typischerweise Grundlage für den Abnahmetest.

Das Pflichtenheft – Tests (2)

Testszenarien sollen die Anforderungen *vollständig abdecken:*

Alle

- Funktionen,
- Daten,
- Leistungen

müssen in wenigstens einem Testfall auftreten!

Auf diese Weise stellen die Testszenarien sicher, dass die gesamte Funktionalität wenigstens einmal getestet wird.

Sehr wichtig: *automatische* Tests, aber schwierig für interaktive Produkte.

Das Pflichtenheft – Entwicklungs-Umgebung

10 Entwicklungs-Umgebung

Software Unix Entwicklungswerkzeuge auf Linux 2.*, speziell Python 2.4, MySQL 4.*, Apache 2.3.

Hardware Opteron 8-fach usw.

Orgware Backup, Versionsverwaltung, Arbeitsplätze, Literatur.

Welche Umgebung wird für die Entwicklung benötigt?

Das Pflichtenheft – Ergänzungen

11 Ergänzungen

Hierunter fallen

- Spezielle, nicht abgedeckte Anforderungen
- Installationsbedingungen
- Zu berücksichtigende Normen, Lizenzen, Patente

Das Pflichtenheft – Glossar

Glossar

Definition aller wichtigen Begriffe zur Sicherstellung einer einheitlichen Terminologie

Glossar soll nicht Allgemeinplätze definieren (etwa „CPU“, „Java“), sondern *Begriffe aus dem Anforderungsbereich*.

Das Pflichtenheft – Glossar (2)

Ein Glossar ist Voraussetzung für sprachliche Präzision:

> *Die Ausleuchtung bestimmt, wie die Weiche dargestellt wird. Ist die Darstellung „Alarm“, so wird die Weiche rot blinkend gezeichnet. Die Abbildung richtet sich nach der Auflösung des Bildschirms.*

Was ist der Unterschied zwischen „Ausleuchtung“, „Darstellen“, „Abbilden“, „Zeichnen“?

Mit zwei Glossar-Einträgen „Ausleuchtung“ und „Darstellung“ wird der Text sofort klarer:

> *Die Ausleuchtung bestimmt, wie die Weiche dargestellt wird. Ist die **Ausleuchtung** „Alarm“, so wird die Weiche rot blinkend **dargestellt.** Die **Darstellung** richtet sich nach der Auflösung des Bildschirms.*

Struktur

Beispiel zu *Produkt-Funktionen*:
Graphischer Editor informal spezifiziert

> To assist in the positioning of entities on a diagram, the user may turn on a grid in either centimetres or inches, via an option on the control panel. Initially, the grid is off. The grid may be turned on and off at any time during an editing session and can be toggled between inches and centimetres at any time. A grid option will be provided on the reduce-to-fit view but the number of grid lines shown will be reduced to avoid filling the smaller diagram with grid lines.

Es geht aber besser (= strukturierter)!

Stark durchgegliederter Text

> **2.6. The Grid**
>
> 2.6.1. The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window. This grid shall be a passive rather than an active grid. This means that alignment is entirely the responsibility of the user and the system should not attempt to align diagram entities with grid lines.
>
> *Rationale:*
> A grid helps the user to create a tidy diagram with well-spaced entities. Although an active grid can be useful, the user is the best person to decide on where entities should be positioned.
>
> 2.6.2. When used in 'reduce-to-fit' mode, the spacing of grid lines shall be adjusted so that line spacing is increased.
>
> *Rationale:*
> If line spacing is not increased, the background will be very cluttered with grid lines.
>
> *Specification*: ECLIPSE/Workstation Tools/DE/FS. Section 2.6.

Stark durchgegliederter Text (2)

Beschreibung und Begründung aller Funktionen

Pflichtenhefte für große Projekte können mehrere hundert bis tausend Seiten umfassen!

3.5.1. Adding nodes to a design

3.5.1.1. To add a node, the editor user selects the appropriate node type from the entity type menu. He or she then moves the mouse so that the cursor is placed within the drawing area. On entering the drawing area, the cursor shape changes to a circle.

Rationale:

The editor must know the node type so that it can draw the correct shape and invoke appropriate checks for that node. The cursor shape change indicates that the editor is in 'node drawing mode'.

3.5.1.2. The cursor should be moved to the approximate node position any mouse button pressed and held down. The node symbol, in a standard size set up by the symbol definer, should appear surrounding the cursor. The symbol may then be dragged, keeping the mouse button depressed, to its final position. Releasing the mouse button fixes the node position and highlights the node.

Rationale:

The user is the best person to decide where to position a node on the diagram. This approach gives the user direct control.

3.5.1.3. If the entity type may be represented using variable-sized symbols, the node highlighting should distinguish this from fixed-size symbols.

Specification: ECLIPSE/Workstation Tools/DE/FS. Section 3.5.1

Produkt-Modellierung

Das Pflichtenheft umfasst gewöhnlich die Szenarien aus den Use-Cases.

Use Cases sind jedoch informal ⇒ Zentrale Konzepte des Systems sollten formaler strukturiert und modelliert werden.

Je nach *Sicht* auf das Gesamt-System können wir folgende Aspekte modellieren:

Funktionen	*Daten*	*Algorithmen*
Regeln	*Zustände*	*Objekte*
	Szenarien	

Grundidee: Die Modelle können sowohl vom *Programmierer* als auch vom *Benutzer* verstanden werden – und so Bestandteil des Pflichtenheftes werden.

Produkt-Modellierung – Basiskonzepte

Funktionale Modellierung Beschreibt die funktionale Hierarchie und den Informationsfluß.

Konzepte: *Funktionsbaum*, *Datenflußdiagramm*

Datenorientierte Modellierung Beschreibt die Datenstrukturen und deren Beziehungen.

Konzepte: *Syntax-Diagramm*, *Entity-Relationship-Diagramm*, *Data Dictionary*

Algorithmische Modellierung Beschreibt das Verhalten in Form von Kontrollstrukturen.

Konzepte: *Pseudocode*, *Struktogramm*, *Jackson-Diagramm*

Erst in der Implementierungsphase!

Produkt-Modellierung: Basiskonzepte (2)

Regelbasierte Modellierung Beschreibt das Verhalten in Form von Wenn-Dann-Regeln.

Konzepte: *Regeln*, *Entscheidungstabelle*

Zustandsorientierte Modellierung Beschreibt die Zustände und die Übergänge.

Konzepte: *Endlicher Automat*, *Petri-Netz*

Objektorientierte Modellierung Beschreibt die auftretenden Klassen und Objekte und deren Beziehungen.

Konzepte: *Klassendiagramm*

Szenariobasierte Modellierung Beschreibt die Interaktion zwischen Komponenten (= Objekten).

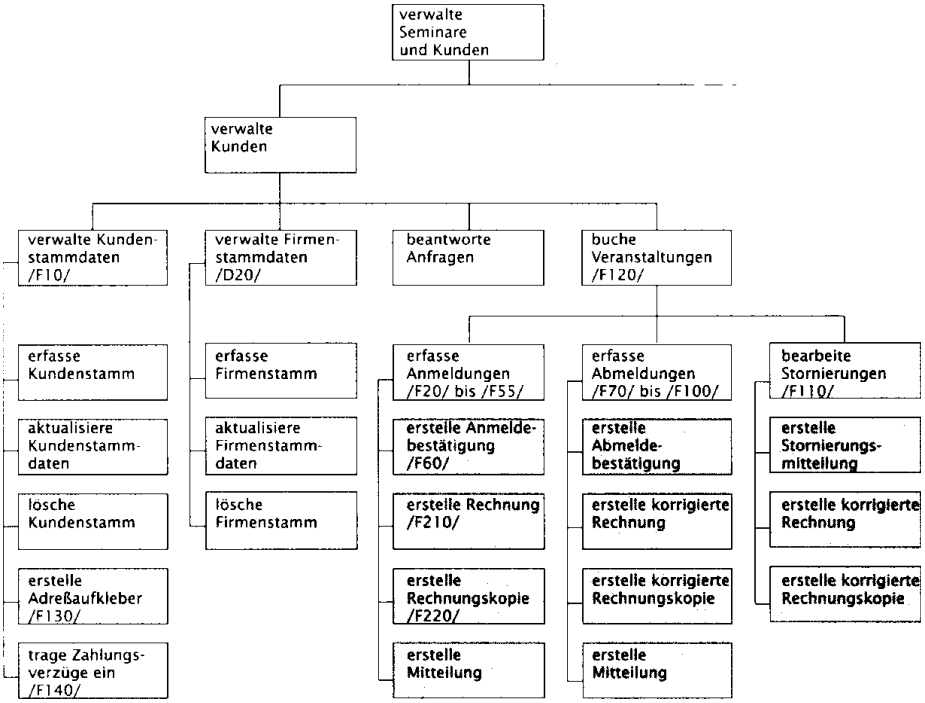
Konzepte: *Interaktionsdiagramm*

Funktionsbaum

Bewährtes Konzept zur systematischen Gliederung von Funktionen

Gibt erste Hinweise für die Dialoggestaltung

Funktionsbaum: *Kundenverwaltung*



Funktionsbaum: Eigenschaften

Der Funktionsbaum berücksichtigt nur die funktionale Sicht; Daten werden ignoriert.

Als *Top-Down*-Methode unterstützt der Funktionsbaum nicht das Finden geeigneter Abstraktionen, was zu *mehrfacher Implementierung* führen kann.

Im Beispiel könnten etwa die Verwaltung von Kundenstamm und Firmenstamm vereinheitlicht werden.

Datenflussdiagramm

Beschreibt, welche Informationen von wo nach wo durch das System fließen

Notation (nach deMarco, 1979): *Structured Analysis*

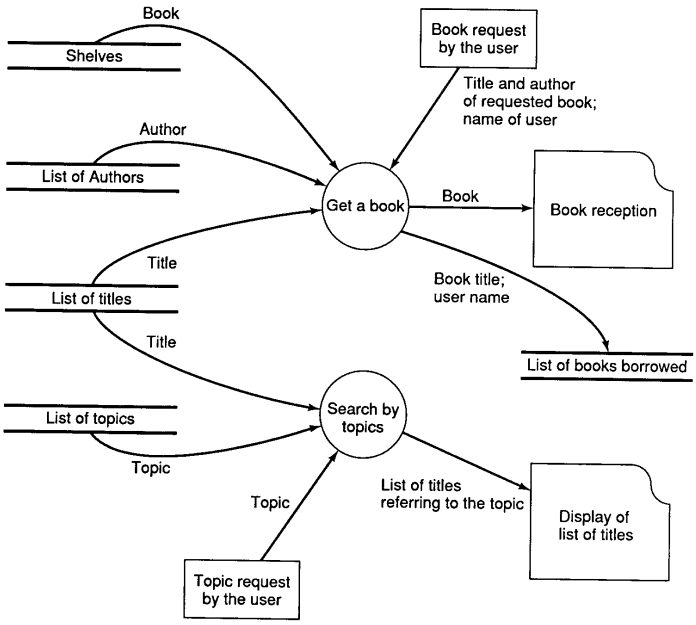
Geschlossene Rechtecke sind *Schnittstellen zur Umwelt* (Informationsquellen und/oder -senken)

Pfeile sind *Datenflüsse* zwischen Schnittstellen, Funktionseinheiten und Datenspeichern

Kreise sind *Funktionseinheiten*, die ankommende Datenflüsse in ausgehende Datenflüsse verwandeln

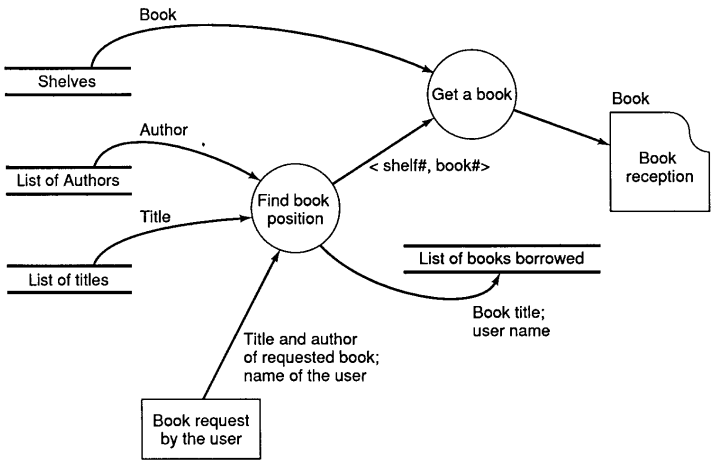
Offene Rechtecke sind *Datenspeicher*, auf die lesend und schreibend zugegriffen werden kann

Datenflussdiagramm: Bibliothekssystem[2]



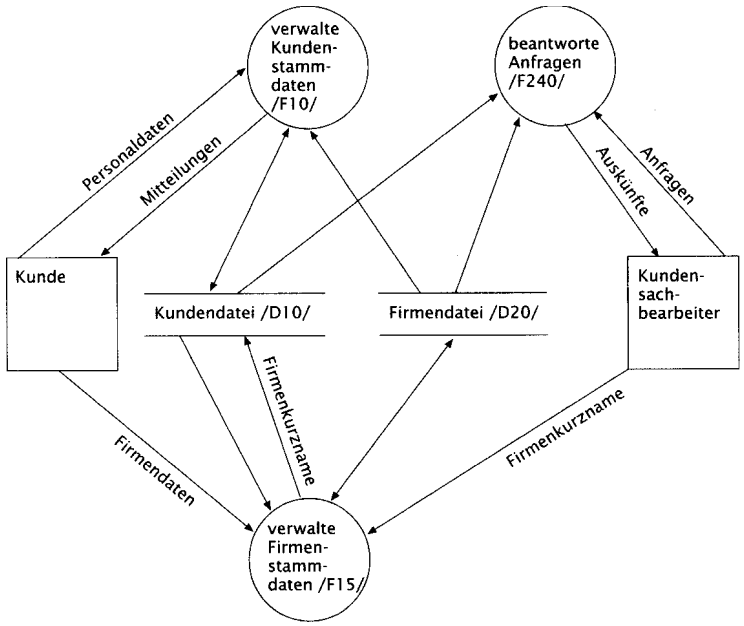
[2] aus Ghezzi et al., Fundamentals of Software Engineering

Bibliothekssystem: Verfeinerung



Datenflüsse von Grobdiagrammen müssen in verfeinerten Diagrammen erhalten bleiben!

Datenflussdiagramm: Kundenverwaltung[3]



[3] aus Balzert, Lehrbuch der Software-Technik

Entity-Relationship-Diagramm

Ein Entity-Relationship-Diagramm (ER-Diagramm) beschreibt die *permanent gespeicherten Daten* und *ihre Beziehungen (datenorientierte Modellierung)*.

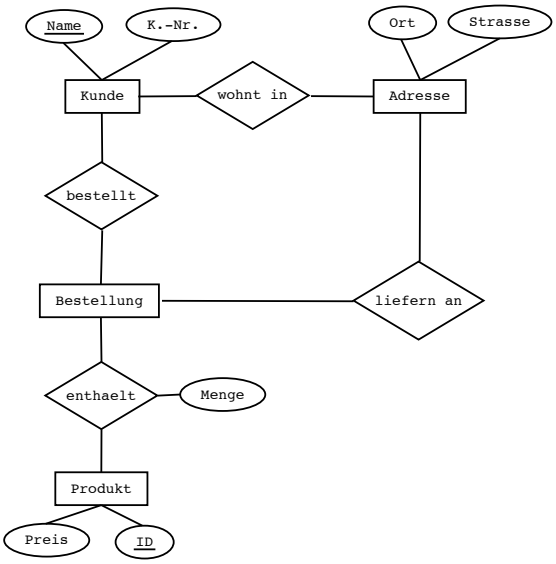
Rechteck: *Entitätsmenge*
Menge von Objekten (unterscheidbar durch Eigenschaften)

Oval: *Attribut*
eine gemeinsame Eigenschaft aller Objekte einer Entitätsmenge

Raute: *Assoziation*
Menge gleichartiger *Beziehungen* zwischen Entitäten

Doppelraute: *Vererbung*
eine Entitätsmenge „erbt“ alle Attribute der bezogenen Entitätsmenge

ER-Diagramm: Beispiel

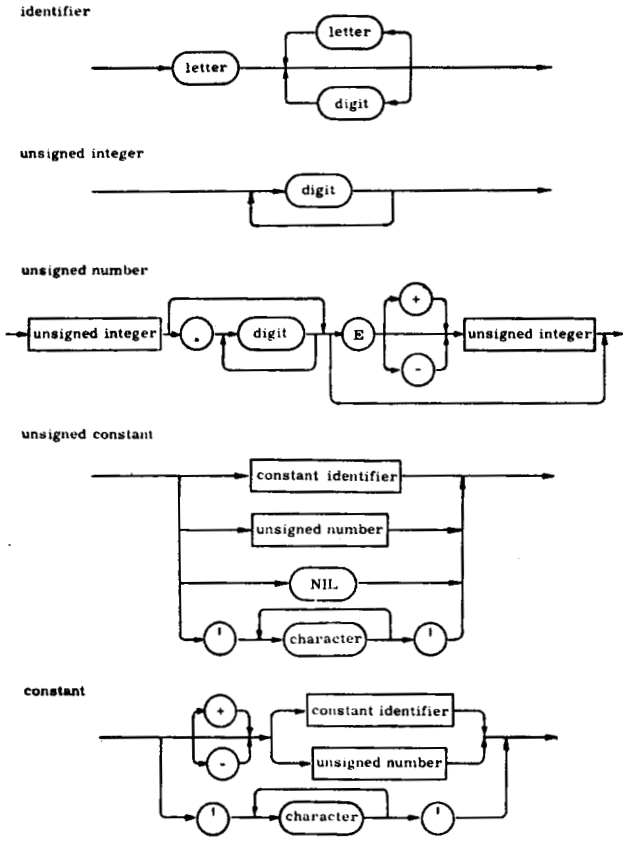


Syntax-Diagramme

Die lexikalischen Einheiten einer Sprache (z.B. der gespeicherten Daten) werden durch *reguläre Ausdrücke* beschrieben

Beispiel: *Bezeichner und numerische Konstanten in Pascal*[a]

[a] aus K. Jensen, N. Wirth, Pascal Manual and Report



Reguläre Ausdrücke

Reguläre Ausdrücke können auch textuell notiert werden.

Es bedeuten:

Folge $X\,Y$ erst X, dann Y

Alternative $(X|Y)$**:** entweder X oder Y

Wiederholung X^***:** 0 oder mehr Auftreten von X

Wiederholung $X^+ = X\,X^*$**:** 1 oder mehr Auftreten von X

Option $X^? = (X|\ )$**:** 0 oder 1 Auftreten von X

Reguläre Ausdrücke (2)

Beispiel: *Aufbau von Bezeichnern in textueller Form*

$$
\begin{array}{rcl}
\textit{identifier} & = & \textit{letter}(\textit{letter}|\textit{digit})^* \\
\textit{unsigned-integer} & = & \textit{digit}^+ \\
\textit{unsigned-number} & = & \textit{unsigned-integer}(\texttt{.}\,\textit{digit}^+)^? \\
 & & (\texttt{E}(\texttt{+}|\texttt{-})^?\,\textit{unsigned-integer})^? \\
\textit{unsigned-constant} & = & \textit{constant-identifier}|\textit{unsigned-number}| \\
 & & \texttt{NIL}|\texttt{'}\,\textit{character}^+\,\texttt{'} \\
\textit{constant} & = & ((\texttt{+}|\texttt{-})^?(\textit{constant-identifier}|\textit{unsigned-number}))| \\
 & & \texttt{'}\,\textit{character}^+\,\texttt{'}
\end{array}
$$

Aus solchen Beschreibungen lassen sich automatisch *endliche Automaten* konstruieren, die Bestandteile einer Eingabe erkennen und verarbeiten.

Mehr dazu in *Theoretische Informatik* und *Compilerbau*.

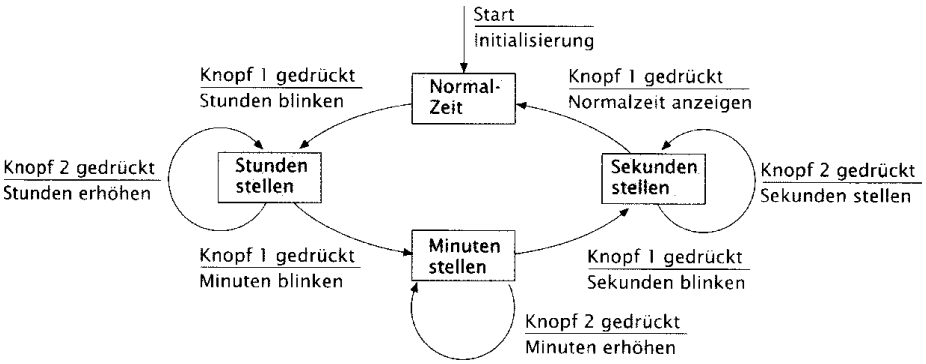
Endliche Automaten

Ein *endlicher Automat* beschreibt ein System als eine (endliche) Menge von *Zuständen (zustandsorientierte Modellierung)*.

Die *Übergänge* zwischen den einzelnen Zuständen sind an bestimmte *Bedingungen* geknüpft.

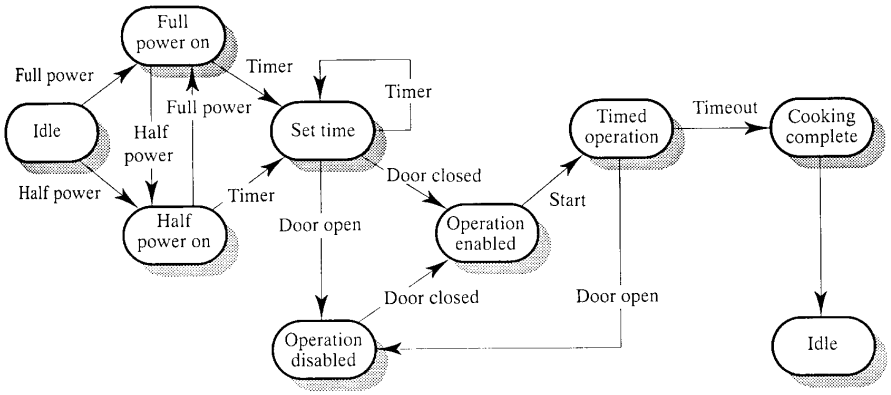
Ist die Bedingung für den Übergang erfüllt, geht das System in einen neuen Zustand über.

Stellen einer Digitaluhr[4]



Mikrowellenofen[5]

Zustände bzw. Übergänge entsprechen realen Schaltern/Sensoren/Lampen



[4] aus Balzert, Lehrbuch der Software-Technik
[5] aus Sommerville, Software Engineering

Mikrowellenofen (2)

Zustände und Übergänge müssen separat erläutert werden:

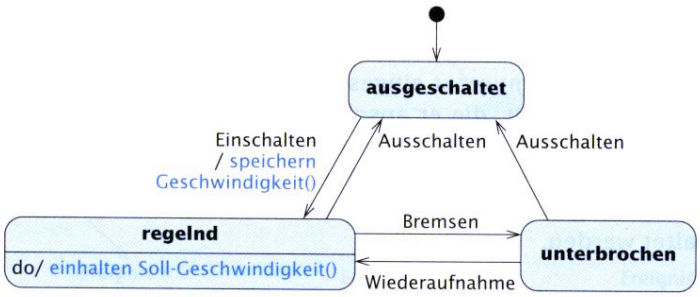
State	Description
Half power on	The oven power output is set to 300 watts
Full power on	The oven power is set to 600 watts
Set time	The cooking time is set to the user's input value
Operation disabled	Oven operation is disabled for safety. Interior oven light is on
Operation enabled	Oven operation is enabled. Interior oven light is off
Timed operation	Oven in operation cooking for the required time. Interior oven light is on
Cooking complete	Timer has reached zero. Sound audible signal. Oven light is off

Stimulus	Description
Half power	The user has pressed the half power button
Full power	The user has pressed the full power button
Timer	The user has pressed one of the timer buttons
Door open	The oven door is not sealed
Door closed	The oven door is sealed
Start	The user has pressed the start button
Timeout	The timer indicates that the set time has expired

Tempomat

Beispiel: *Feinstruktur eines Tempomats*[6]

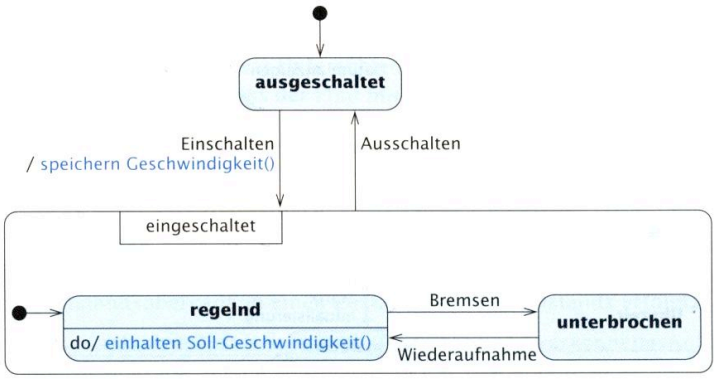
Dieses Beispiel illustriert, wie Automaten *abstrahiert* werden können, indem ein Teilautomaten zu einem Zustand wird. Die umgekehrte Richtung, also Verfeinerung, ist ebenfalls möglich.



Natürlich müssen die ursprünglichen Übergänge aus/in einen verfeinerten Zustand sich im neuen Teilautomaten wiederfinden!

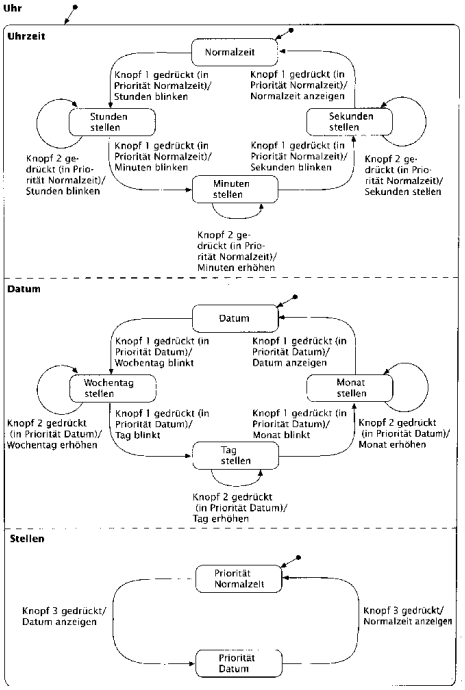
[6] aus Balzert, Lehrbuch der Softwaretechnik

Tempomat (2)



Statecharts

Erlauben die Beschreibung *nebenläufiger* Zustände



Petri-Netze

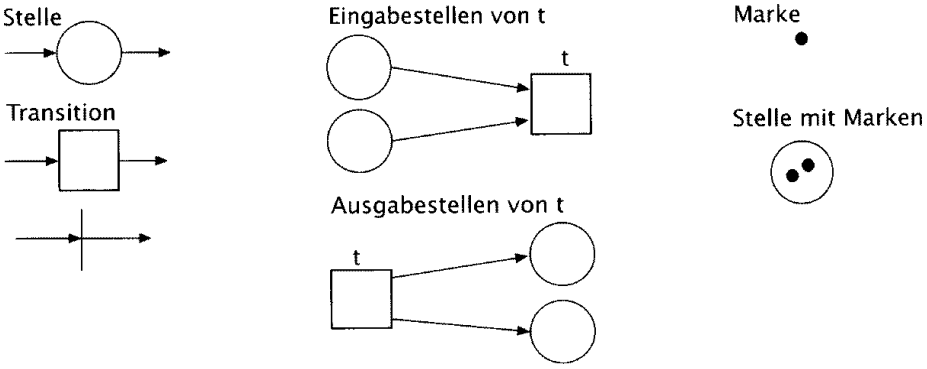
Beschreiben *parallele* und *asynchrone* Systeme

Petri-Netze sind gerichtete Graphen, deren Knoten entweder *Stellen* oder *Transitionen* sind

Stellen dienen als Aufnahmebehälter für *Marken*

Transitionen sind die Übergabewege für Marken

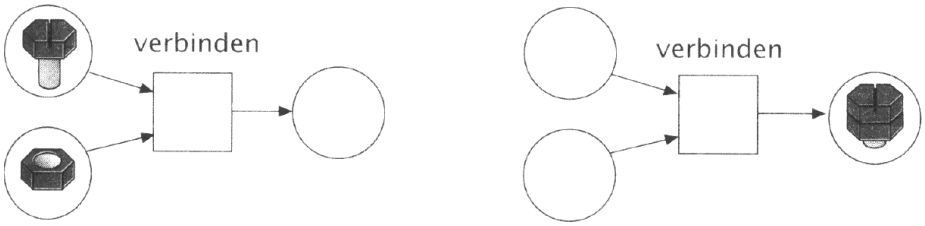
Marken beschreiben durch ihre Anzahl und Position den Zustand eines Petri-Netzes



Petri-Netze (2)

Der Bewegungsablauf von Marken im Netz wird durch die *Schaltregel* festgelegt:

- Eine Transition t kann schalten, wenn jede Eingabestelle von t wenigstens eine Marke enthält
- Schaltet eine Transition, wird aus jeder Eingabestelle eine Marke entfernt und jeder Ausgabestelle eine Marke hinzugefügt



Bedingungs/Ereignis-Netz

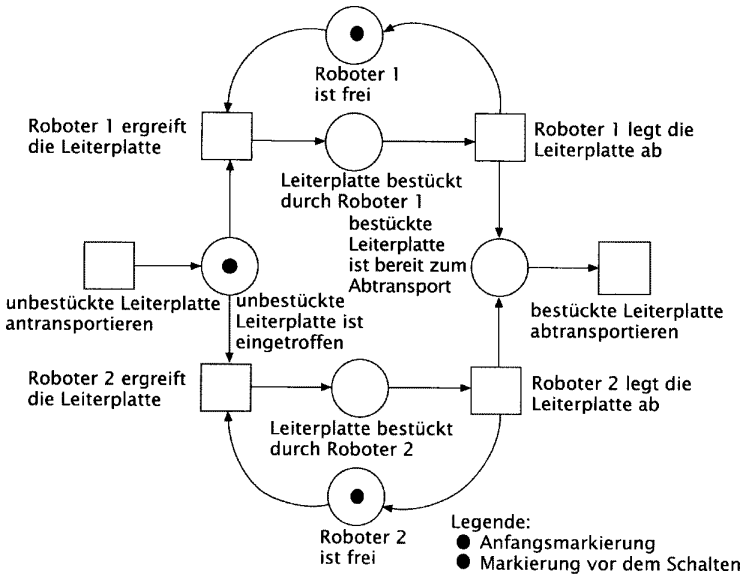
Einfachste Form von Petri-Netzen: Jede Stelle kann genau eine oder keine Marke enthalten

Weitere Schaltregel:

- Eine Transition t kann schalten, wenn jede Eingabestelle von t eine Marke enthält und jede Ausgabestelle von t leer ist

Leiterplatten bestücken

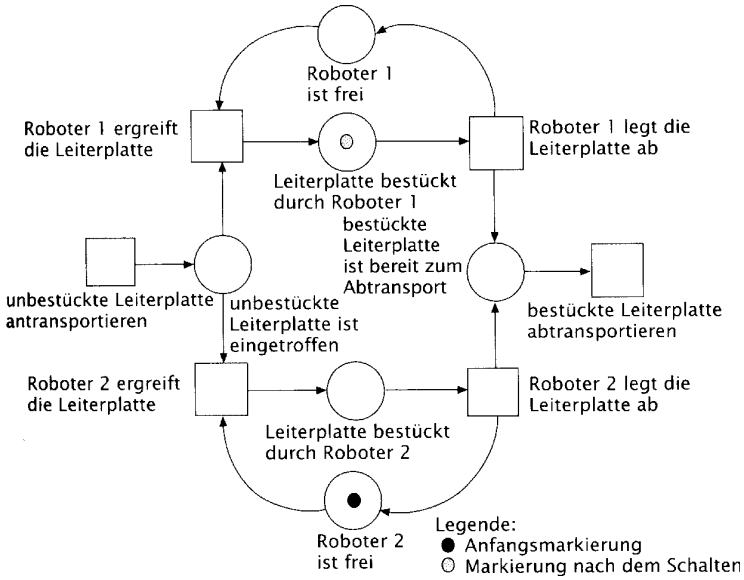
Beispiel: *Bestücken von Leiterplatten durch zwei Roboter*[7]



Welche der beiden Transitionen feuert (und welcher Roboter die Leiterplatte ergreift), ist nicht deterministisch!

[7] aus Balzert, Lehrbuch der Software-Technik

Leiterplatten bestücken (2)



Weitere Petri-Netze

Stellen/Transitions-Netze Jeder Kante ist eine maximale *Kapazität* zugeordnet; jede Transition muß entsprechend der Kapazität Marken hinzufügen und entfernen

Prädikat/Transitions-Netze Marken sind unterschiedlich *gefärbt*; *Prädikate* an den Transitionen geben an, unter welchen Bedingunen die Transitionen schalten dürfen

Hierarchische Petri-Netze zur Strukturierung großer Systeme

Zeitbehaftete Petri-Netze mit „Verharren“ der Marken auf Transitionen

Entscheidungstabellen

Entscheidungstabellen sind eine *Matrix* zur Festlegung von unter bestimmten Bedingungen auszuführenden Aktionen *(regelbasierte Modellierung).*

Obere Hälfte der Matrix: *Bedingungsteil.* Die Zeilen sind mit Bedingungen markiert

Untere Hälfte der Matrix: *Aktionsteil.* Die Zeilen sind mit auszuführenden Aktionen markiert

Jede Spalte entspricht einer Bedingungskombination

Entweder nur ja/nein Einträge oder differenzierte Einträge

Bibliothekssystem[8]

Eingegangene Publikationen \ Regeln	1	2	3	4	5	6	7	8	9	sonst
Art des Eingangs	A	A	A	A	G	G	G	G	B	
In "Bestellt"-Kartei registriert ?	J		N	N	J		N	N	J	
Schon vorhanden ?		J	N	N		J	N	N		
Paßt zum Bestand ?			N	J			N	J		
Bestellen und in "Bestellt"-Kartei registr.										
Rechnung zur Bearbeitung weiterleiten				X					X	
Eintrag in "Bestellt"-Kartei löschen									X	
Zurücksenden	X	X	X							
Aussondern (Einstampfen oder Verschenken)							X			
Zur Titelaufnahme weiterleiten				X	X			X	X	
An Sonderfallbearbeitung weiterleiten					X	X				X

Erläuterung der Abkürzungen:

A - zur Ansicht
B - bestellt
G - Geschenk
J - Ja
N - Nein

Entscheidungstabellen – Eigenschaften

Vollständigkeit (jede Bedingungskombination muß abgedeckt sein) kann automatisch geprüft werden

Widerspruchsfreiheit (es dürfen keine sich widersprechenden Aktionen ausgelöst werden)

[8] aus Kimm et al. Einführung in Software Engineering

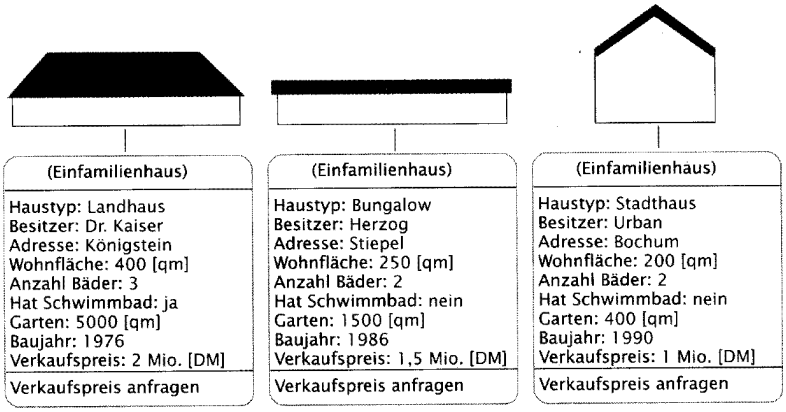
Kann nicht automatisch geprüft werden, denn ein Rechner weiß nichts über die Semantik der Aktionen

Aus einer Entscheidungstabelle kann automatisch ein Programm(gerüst) erzeugt werden.

Objektorientierte Modellierung

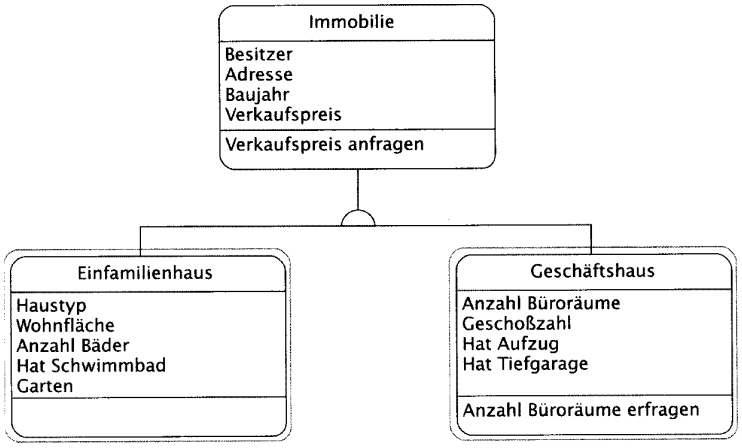
Beschreibt ein System mit Hilfe von *Objekten*, die anhand ihrer Eigenschaften in *Klassen* zusammengefaßt werden.

Neben Eigenschaften kennen Objekte auch *Methoden*: Funktionen, die sich implizit auf das jeweilige Objekt beziehen.



Klassenhierarchie

Die *Klassenhierarchie* drückt über *Vererbung* gemeinsame Eigenschaften und Methoden aus:



Detaillierte Beschreibung später („Entwurf“)

Szenariobasierte Modellierung

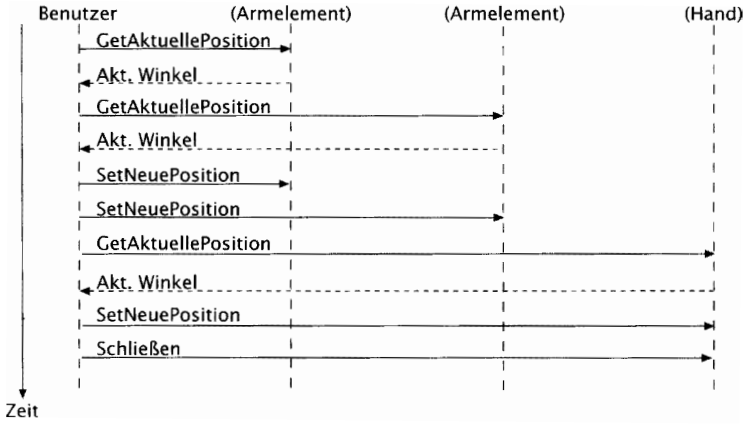
Dient der schematischen Veranschaulichung *zeitbasierter Vorgänge*

Wird in der objektorientierten Modellierung benutzt, um die *Kommunikation zwischen Objekten* darzustellen

Hilfreich zur Darstellung von *Szenarien* und *nebenläufiger Prozesse*

Robotersteuerung

Ein Benutzer positioniert einen Roboter mit Kommandos[9]



Zusammenfassung

- ✔ In der herkömmlichen Entwicklung werden Use-Cases zu einem *Pflichtenheft* weiterentwickelt, das Vertragsgrundlage wird.
- ✔ Das Pflichtenheft beschreibt die Funktionen und Leistungen des Systems.
- ✔ Zentrale Aspekte des Systems werden (semi)-formal strukturiert und modelliert; die Modelle werden dabei vom Programmierer als auch vom Benutzer verstanden.
- ✔ Verbreitete Modellierungsverfahren gibt es für Funktionen, Daten, Algorithmen, Regeln, Zustände, Objekte, und Szenarien.

[9] aus Balzert, Lehrbuch der Software-Technik