

Andere Analysemethoden

W. Henhapl

- SA
 - Methodik
 - Kontext-Diagramm
 - Data Flow Diagramme
 - Data Dictionary
 - ER-Diagramme
 - Mini-Spezifikationen

- Formale Methoden
 - Anforderungen
 - Modellorientiert
 - Algebraisch

Übersicht:

- Beschreibung durch Kombination von Modellen
 - Context Diagram entspricht UC-Diagramm für Systemgrenze
 - Dataflow Diagram als zentrales Beschreibungsmittel für die Systemstruktur. Zerlegungseinheiten sind Funktionen. Zweck UC hierarchisch zu verfeinern.
 - Data Dictionary zur Beschreibung der Datenflüsse
 - Entity Relationship Diagram zur Beschreibung der Entities
 - Mini-Specifications
 - Pseudosprachen
 - Pre/Post conditions
 - State Transition Diagrams
 - Decision Tables
 - Flowcharts, Nassi-Shneiderman Diagrams

Ref: <http://www.yourdon.com/books/msa2e/>

- Methode
 - Start: Feature List und Vision
 - 1. Schritt: Context Diagramm, Entity Relationship Diagramm externer DBSs
 - i-ter Schritt:
 - Verfeinerung eines Prozesses und des entsprechenden Teils des Data Dictionarys oder
 - Definition des Prozesses als Mini-Specification
 - Ende: Alle Prozesse sind verfeinert oder spezifiziert
- Überzeugung:
 - Dataflow Diagramme sind intuitiv und könne daher vom Benutzer validiert werden
 - Mini-Specification abhängig von Benutzer wählen
- Trivial Implementierung
 - Dataflows z.B. in C:
 - globale Variable
 - Struktur direkt aus Data Dictionary ableitbar
 - Prozesse als Prozeduren

The **boundary** between system and the rest of the world.

Actors: The people, organizations, or systems with which our system communicates. These are known as *terminators*.

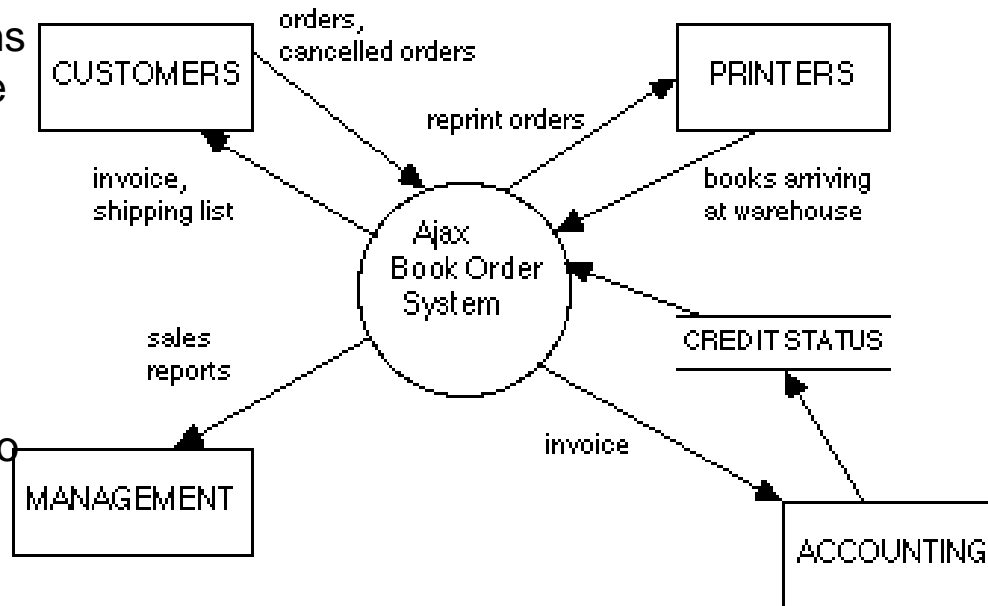
Data Communication:

The data that our system receives from the outside world and that must be processed in some way.

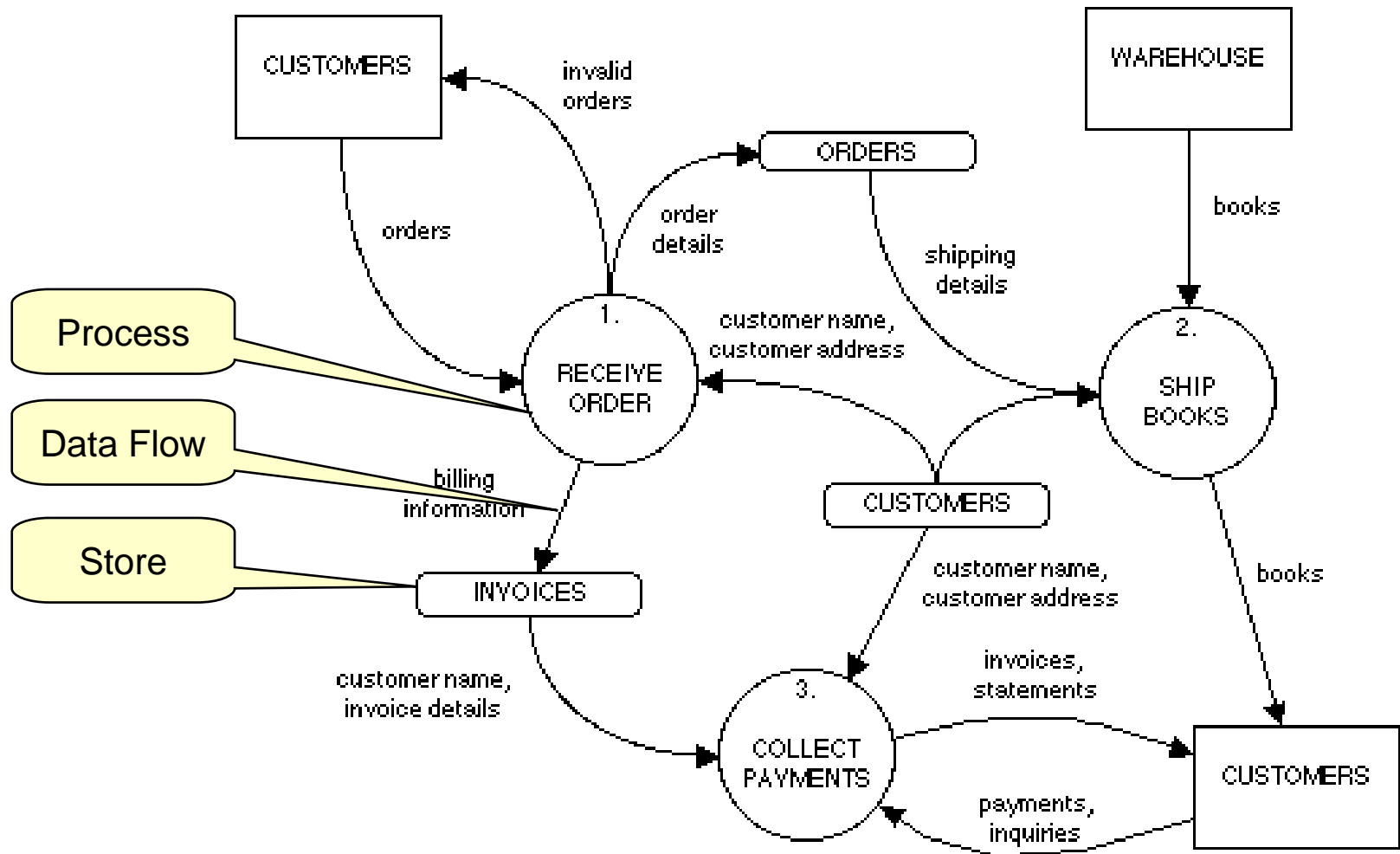
The data produced by our system and sent to the outside world.

Persistent Data:

The data stores that are shared between our system and the terminators. These data stores are either created outside the system and used by our system or created by our system and used outside the system.

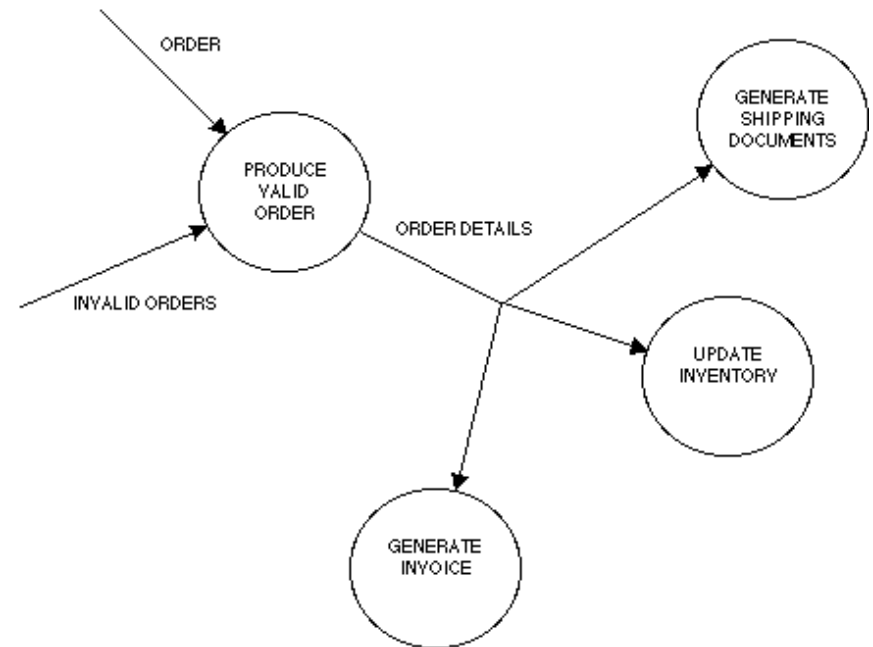


SA: Data Flow Diagram



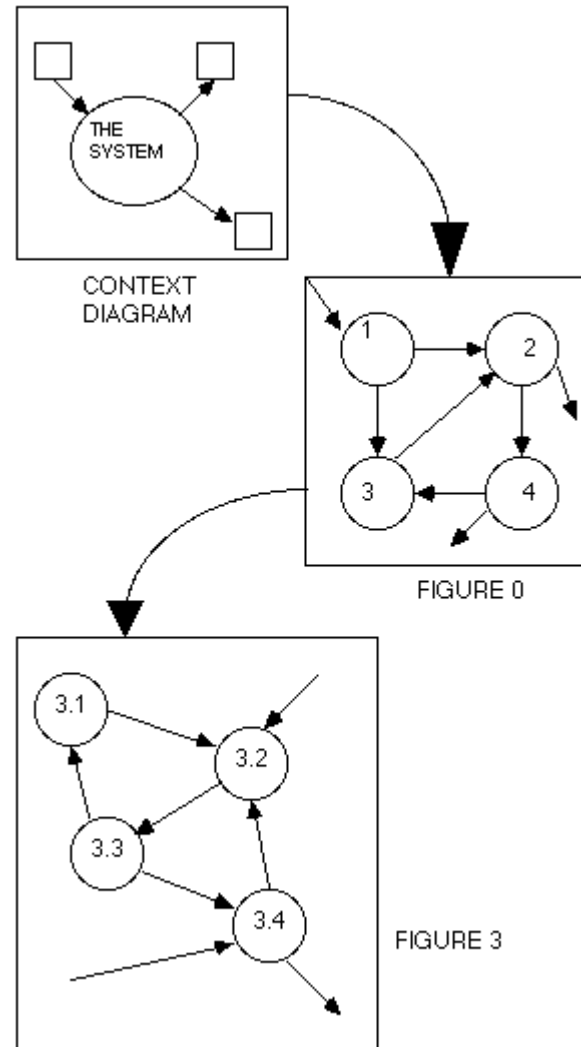
Data Flow Diagram

- Data flow:
 - Prompt or request
 - Type of data in Data Dictionary
- Process:
 - No execution order. Process can start when all incoming data are available
 - No execution time
 - Only logical ordering by the data streams



SA: Hierarchical Data Flow Diagramms

- Stepwise Refinement
- Consistent Refinement



Data description as a grammar:

- = is composed of
- + and
- () optional (may be present or absent)
- { } iteration
- [] select one of several alternative choices
- ** comment
- @ identifier (key field) for a store
- | separates alternative choices in the [] construct
- Elementary Data

name = courtesy-title + first-name + (middle-name) + last-name

courtesy-title = [Mr. | Miss | Mrs. | Ms. | Dr. | Professor]

first-name = {legal-character}

middle-name = {legal-character}

last-name = {legal-character}

legal-character = [A-Z|a-z|0-9|'|-'| |]

Elementary Types:

current-weight = int

**

“units: pounds; range: 1-400*”

current-height = int

**

units: inches; range: 1-96

date-of-birth = int

**

units: days since Jan 1, 1900; range: 0-36500

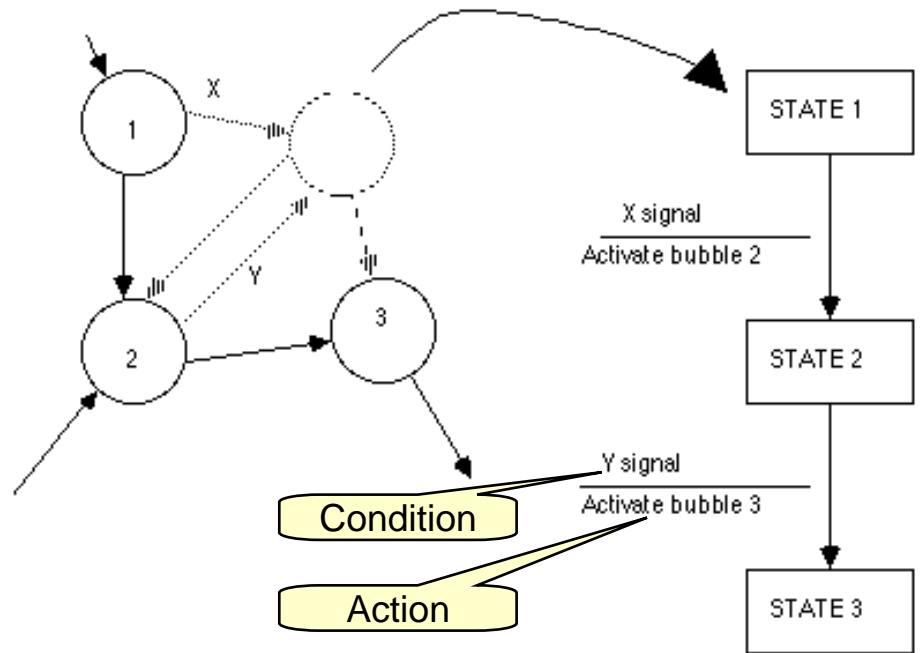
sex = [M | F]*

Additional Information:

1. The *meaning* of the data element within the context of this user's application. This is usually provided as a comment, using the “**” notation.
2. The *composition* of the data element, if it is composed of meaningful elementary components.
3. The legal *values* that the data element can take on, if it is an elementary data element that cannot be decomposed any further.

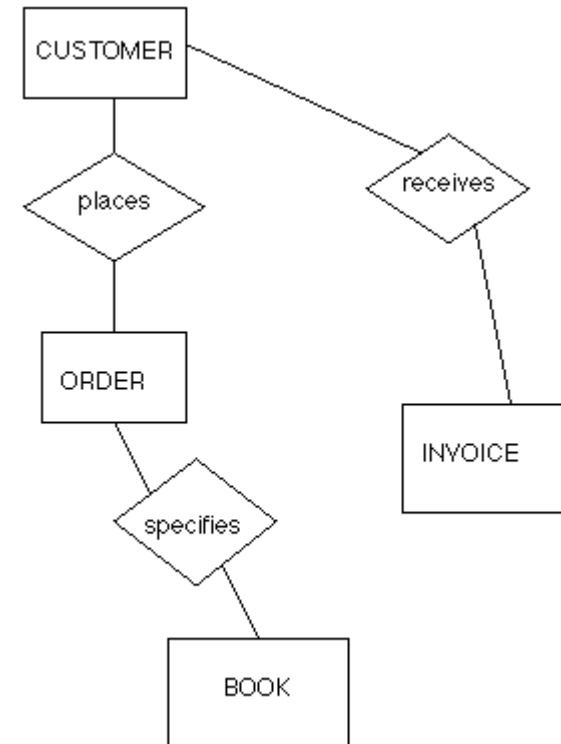
State Transition Diagram

- Subset of UML



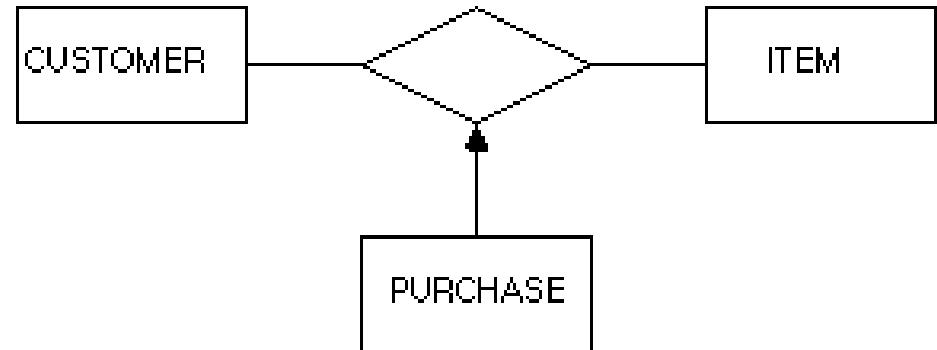
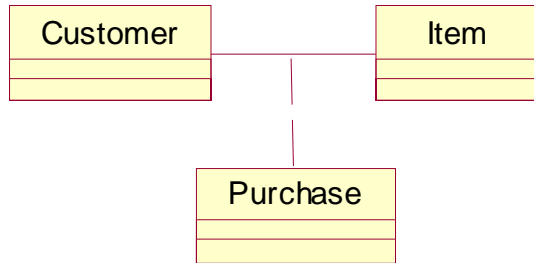
SA: Entity Relationship Diagram

- For communication with the DB-Administration group
- Elements
 - Object types (type with attributes) (entity classes without methods in UML)
 - Relationships (associations in UML)
 - Associative object type indicators (association class in UML)
 - Supertype/subtype indicators (Generalisation in UML)



SA: Entity Relationship Diagram

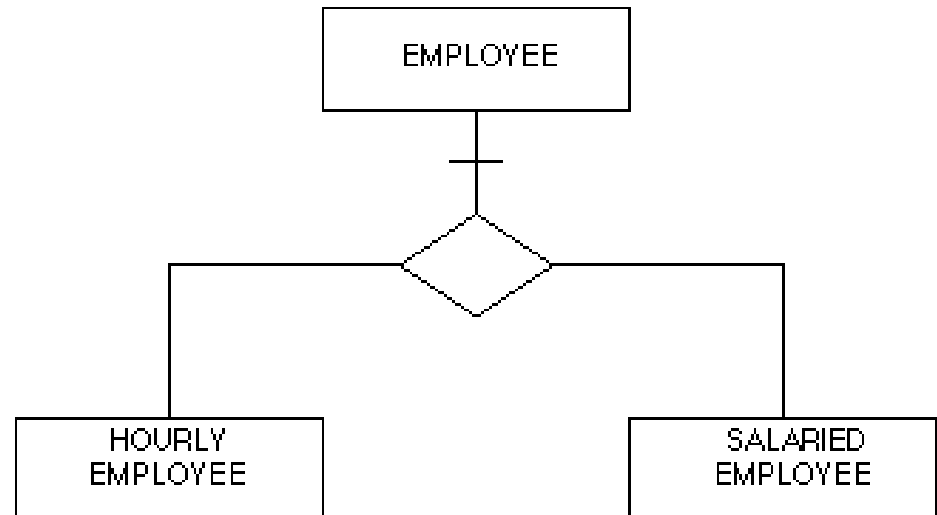
- Associative object type indicators:
 - Relation has its own Attributes



- Extension of Data Dictionary for ERD

CUSTOMERS = {**CUSTOMER**}

CUSTOMER = @customer-name + address + phone-number



Process takes some actions based on *complex decisions*.

- Possible States:
 - Above the heavy line
 - Collection of all relevant combinations (rules) of state variables
 - Normally binary values
 - Don't care for reducing the number of combinations
- Actions:
 - Under the heavy line
 - Actions should be independent if they occur in one rule

	1	2	3	4	5	6	7	8
Age > 21	Y	Y	Y	Y	N	N	N	N
Sex	M	M	F	F	M	M	F	F
Weight > 150	Y	N	Y	N	Y	N	Y	N
Medication 1	X				X			X
Medication 2		X			X			
Medication 3			X			X		X
No medication				X			X	

Qualitätsmerkmale formaler Spezifikationen

- **Eindeutigkeit:**
 - Antworten bezüglich des Verhaltens sind immer eindeutig
 - Antworten haben immer eine und nur eine Interpretation
- **Vollständigkeit:** Jede Frage kann beantwortet werden!
- Soll ein Use Case Model eindeutig und vollständig sein?

Ist Vollständigkeit überhaupt erforderlich?

- Nur **public Funktionen** beantworten Fragen!
- Hinreichend vollständig: Jede durch die Schnittstelle **zugelassene** Frage kann beantwortet werden.
- Daher strikte Trennung zwischen Funktionen und Zustandsänderungen!
- Hinreichend vollständig:
 - Fragen haben die Form: (Zustandstransformation)* Funktion

- Rückblick
 - Requirements:
 - Feature list
 - Use Case Model
 - Architectural Description
 - Analysis
 - Analysis Model
- } informal
- halbformal
-
- Vorteile?
 - Lesbar für verschiedene Interessenten
 - Daher offen für Diskussion und Konsens
 - ?
-
- Nachteile:
 - Vage Semantik
 - Prüfbarkeit:
 - Eindeutigkeit?
 - Vollständigkeit? Darf das geplante System nur die definierten Use Cases zulassen?
 - Widerspruchsfreiheit?
 - Geringe Werkzeugunterstützung, abhängig von der Formalisierung

Daher formale Spezifikation früher Teil von SE

- Datenströme
 - AS1
 - Edifac
 - XML
 - ...
- Telekommunikation / Elektrotechnik
 - State Charts
 - Entscheidungstabellen
 - SDL, Lotos, Petrinetze
 - ...
- Sicherheitsrelevanz
 - Pre- und Post Conditions
 - Abstrakte Datentypen (Algebraische Spezifikation)
 - ...

- Syntaxbeschreibungen
 - Semantikbeschreibungen
 - Operational:
 - Beispiele:
 - Endliche Automaten (State Charts, ...)
 - Funktionale Beschreibungen
 - Schließen?
 - Anwendung von (bewiesenen) Transformationen bis zur Implementierung
 - Twin-Machines zum Beweis der Äquivalenz
 - Problem: Was ist Äquivalenz?
 - Deskriptive Beschreibung notwendig!
- Deskriptiv:
 - Vor- und Nachbedingungen (VDM, Z, OCL, ...)
 - Modellorientierte Spezifikation
 - Algebraische Spezifikationen

- Basierend auf Kontraktkonzept
 - Precondition: Zulässigkeit des Aufrufs mit den gegebenen Argumenten
 - Postcondition: Garantie der spezifizierten Ergebnisse
 - Vorsicht **kein** Bezug zum Hoare Kalkül!
- Warum der Name „modellorientiert“?
 - Notwendigkeit den inneren Zustand zu modellieren bzw. offen zulegen
 - Beispiel: Stack
 - Konstruktor empty
 - Pre: true
 - Post: is-empty
 - push(x)
 - Pre: true
 - Post: not is-empty and top = x
 - pop
 - Pre: not is-empty
 - Post: ???

- Modellangabe: **Intern** Stack als Liste L
 - Konstruktor empty
 - Pre: true
 - Post: is-empty and $L = \langle \rangle$ $\langle \rangle$ leere Liste
 - push(x)
 - Pre: true
 - Post: not is-empty and top = x and $L = \langle x \rangle + \text{old } L$ + Listenzusammensetzung
 - pop
 - Pre: not is-empty
 - Post: $\exists x. \text{old } L = \langle x \rangle + L$
 - is-empty
 - Pre: true
 - Post: $L = \langle \rangle$

- Modellbildung mit Standarddatentypen:
 - Mengen, Listen, Produkttyp, Summentyp, Tabellen, ...
 - Alles + old in OCL vorhanden
 - OCL ist der von der OMG akzeptierte Standard für das Kontraktkonzept

Übung:

1. Spezifikation für den beschränkten Stack
2. Sind die Spezifikationen äquivalent?

Modell: Statt Liste Mengen

- Konstruktor empty
 - Pre: true
 - Post: is-empty and $S = \{\}$
- push(x)
 - Pre: true
 - Post: not is-empty and top = x and $S = \{x, \text{old } S\}$
- pop
 - Pre: not is-empty
 - Post: $\exists x. \text{old } S = \{x, S\}$
- is-empty
 - Pre: true
 - Post: $L = \langle \rangle$

- Nutzen:
 - Redundanz (zur Implementierung) erzwingt wenigstens Nachdenken
 - Testen:
 - Zulässigkeit der Aufrufe: **Unbedingt realisieren. Aber nur mit öffentlichen Methoden**
 - Gegebenes Orakel für das Testen
 - Daher soviel wie möglich in die Post-Condition stecken
- Nachteile:
 - Unklar was wichtig ist oder nur Eigenschaft des Modells
 - Geschwätzig
 - Beweise schwer durchführbar
 - Anforderungen an Spezifikationen nur teilweise erfüllt:
 - Eindeutigkeit: Ja
 - Vollständigkeit: Üblicherweise nein
 - Widerspruchsfreiheit: Ja

- Beschreibung durch „Gleichungen“:
 - Terme die als äquivalent angesehen werden
- Standardtechnik:
 - Strenge Unterscheidung:
 - Operationen
 - Funktionen
 - Hinreichende Vollständigkeit:
 - Für jeden Term dessen „oberster“ Term eine Funktionen ist, muss der Wert des Terms erchenbar sein.
 - Standardtechnik:
 - Auffinden der minimalen Erzeuger (Konstruktoren)
 - Funktionswerte für diese „kanonische“ Terme als Gleichung angeben
 - Jeder weiteren Operation in der Gleichung eine äquivalente Konstruktion über die Erzeuger angeben
 - Beispiel: Stack
 - Erzeuger: empty und push
 - $\text{is-empty(empty)} = \text{true}$ und $\text{is-empty(push(x, s))} = \text{false}$
 - $\text{top(empty)} = \text{error}$ und $\text{top(push(x,s))} = x$
 - $\text{pop(empty)} = \text{error}$ und $\text{pop(push(x,s))} = s$

➤ Übung:

1. Warteschlange: Spezifikation für Signatur `Empty`, `is-empty`, `enqueue`, `dequeue`, `next`
2. Stack ohne `pop(push(x,s)) = s` ? Interpretationen?

- Nutzen:
 - Keine Redundanz. Beweise können direkt erfolgen
 - Testen:
 - Zulässigkeit der Aufrufe: **Unbedingt realisieren. Aber nur mit öffentlichen Methoden**
 - Gegebenes Orakel für das Testen
 - Daher soviel wie möglich in die Post-Condition stecken
- Nachteile:
 - Unklar was wichtig ist oder nur Eigenschaft des Modells
 - Geschwätzig
 - Beweise schwer durchführbar
 - Anforderungen an Spezifikationen nur teilweise erfüllt:
 - Eindeutigkeit: Ja
 - Vollständigkeit: Üblicherweise nein
 - Widerspruchsfreiheit: Ja