Waterfall Model:
system analysis -> software specification -> architecture design -> refined design and coding -> integration and testing -> installation and acceptance -> operation and maintenance (instandhalten)

Waterfall or Document-Model— Software develop-
mentisseenasasequenceofactivitiescoupledby(partial) results (documents). These activities can be conducted concurrently or iteratively. Apart from that, the sequence of activities is fixed as (basically) analyse, specify, design, code, test, install, maintain. Ludewig & Lichter (2013)

Idea of **Spiral Model**: do not plan ahead everything, but go step-by-step.

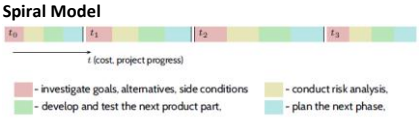
Repeat until end of project (successful completion or failure):

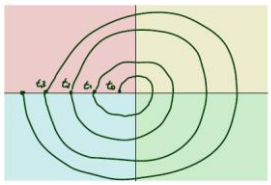
(i) **determine** the set R of **risks** which are **threatening** the project; if $R = \emptyset$, the project is successfully completed

(ii) **assign** each risk $r \in R$ a **risk value** $v(r)$

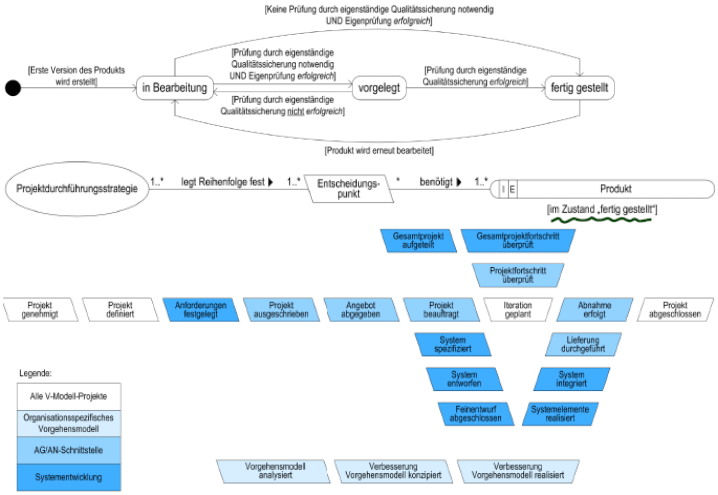
(iii) for the risk r_0 with the **highest risk value**, $r_0 = \max\{v(r) \mid r \in R\}$, find a way to eliminate this risk, and go this way; if there is no way to eliminate the risk, stop with project failure

Spiral Model





V-Modell XT:

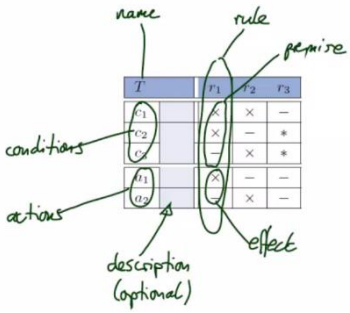


Lines of Code:
program size – LOCtot – number of lines in total; net program size – LOCne – number of non-empty lines; code size – LOCpars – number of lines with not only comments and non-printable; delivered program size – DLOCtot,ne,pars – like LOC, only code (as source or compiled) given to customer

Decision Table

DT: 3	R1	R2	R3
C1	×	–	*
C2	×	*	×
C3	*	*	*
A1	×	–	–
A2	–	–	×
$\neg[C1 \wedge C2]$			

A1 ↯ A2



c1 und c2 können wegen dem *Konfliktaxiom* niemals gleichzeitig true sein. R1 verlangt das aber, daher ist R1 eine *vaccuos rule* (leer).
A1 und A2 dürfen laut *conflikt relation* nicht gleichzeitig aktiviert werden, daher ist Tabelle *konsistent*. Ansonsten *inkonsistent*.
rule ist *useless*, wenn ihre premise und actions von einer anderen rule überdeckt werden. Sie kann entfernt werden.
Tebelle ist *deterministic*, wenn alle premises paarweise disjunkt sind. In dem Fall (links) nicht.
Tabelle ist *complete*, iff disjunction of all rules' premises is a tautology. *Incomplete*: premise finden, die nicht in Tabelle steht.
Decision tables are *correct* (it correctly represents the wishes/needs of the customer), *complete* (all requirements existing should be present), *relevant* (things which are not relevant to the project should not be constrained), *consistent, free of* contradictions (each requirement is compatible with all other requirements; otherwise the requirements are not realizable), *neutral, abstract* (a requirements specification does not constrain the realization more than necessary), *traceable, comprehensible* (the source of requirements are documented, req. are uniquely identifiable), *testable, objective* (the final product con objectively be checked for satisfying a req.)

Use case diagram:

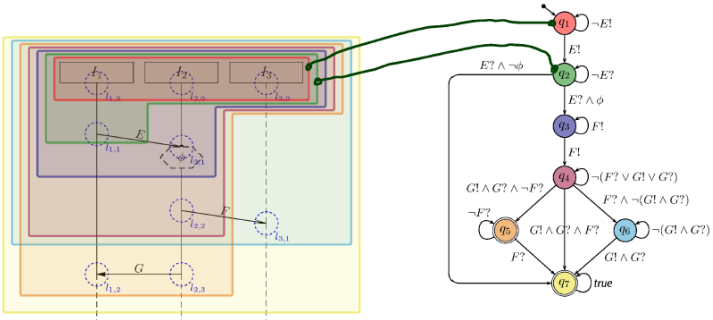
Use Case Example

name	Authentication
goal	the client wants access to the ATM
pre-condition	the ATM is operational, the welcome screen is displayed, card and PIN of client are available
post-condition	client accepted, services of ATM are offered
post-cond. in exceptional case	access denied, card returned or withheld, welcome screen displayed
actors	client (main actor), bank system
open questions	none
normal case	1. client inserts card 2. ATM read card, sends data to bank system 3. bank system checks validity 4. ATM shows PIN screen 5. client enters PIN 6. ATM reads PIN, sends to bank system 7. bank system checks PIN 8. ATM accepts and shows main menu
exception case 2a	card not readable 2a.1 ATM displays "card not readable" 2a.2 ATM returns card 2a.3 ATM shows welcome screen

exc. case 2b	card readable, but not ATM card
exc. case 2c	no connection to bank system
exc. case 3a	card not valid or disabled
exc. case 5a	client cancels
exc. case 5b	client doesn't react within 5 s
exc. case 6a	no connection to bank system
exc. case 7a	first or second PIN wrong
exc. case 7b	third PIN wrong

(Ludewig and Lichter, 2013; V-Modell XT, 2006)
14/49

Live Secuence Charts:
Ein *Cut* ist warm, wenn mind. eine der letzten Locations warm ist. Diese sind warm, wenn die Linie danach durchgezogen ist. Ist ein Cut kalt, ist dieser ein Finalzustand im Büchiautomat.



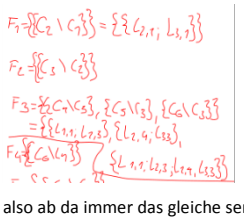
- $\mathcal{L} : l_{1,0} \prec l_{1,1} \prec l_{1,2} \prec l_{1,3},\ l_{1,2} \prec l_{1,4},\ l_{2,0} \prec l_{2,1} \prec l_{2,2} \prec l_{2,3},\ l_{3,0} \prec l_{3,1} \prec l_{3,2},\ l_{1,1} \prec l_{2,1},\ l_{2,2} \prec l_{1,2},\ l_{2,3} \prec l_{1,3},\ l_{3,2} \prec l_{1,4},\ l_{2,2} \sim l_{3,1},$
- $\mathcal{I} = \{\{l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}, l_{1,4}\}, \{l_{2,0}, l_{2,1}, l_{2,2}, l_{2,3}\}, \{l_{3,0}, l_{3,1}, l_{3,2}\}\},$
- $\text{Msg} = \{(l_{1,1}, A, l_{2,1}), (l_{2,2}, B, l_{1,2}), (l_{2,2}, C, l_{3,1}), (l_{2,3}, D, l_{1,3}), (l_{3,2}, E, l_{1,4})\}$
- $\text{Cond} = \{(\{l_{2,2}\}, c_2 \wedge c_3)\},$
- $\text{LocInv} = \{(l_{1,1}, \circ, c_1, l_{1,2}, \bullet)\}$

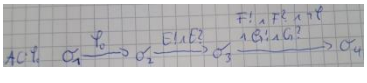
Bei dieser local invariant ist l1,1 nicht eingeschlossen, l1,2 aber schon

LSC: $\mathcal{L}_1$
AC: c_1
AM: initial I: permissive

Wenn full LSC hot (*universal*) ist, muss bei entsprechender activation condition (AC) genau dieses LSC durchlaufen werden (durchgezogener Rand um den LSC body). Ist es cold (*existential*) reicht es, wenn dieses LSC möglich ist.
Wenn activation mode (am) ist *initial* und LSC body ist cold, dann gibt es mind eine Belegung, die von Anfang startet und akzeptiert wird.
am *invariant* und body cold: es gibt mind eine Belegung zu irgendeinem Zeitpunkt, die akzeptiert wird.
am *initial* und body hot: alle Belegungen werden vom Anfangszeitpunkt an akzeptiert. (z.B. Computer hochfahren)
am *invariant* und body hot: Alle Belegungen werden zu irgendeinem Zeitpunkt akzeptiert z.B. an-/ausschalten von Ventilator.
LSC ist *strict*, wenn alle messages, die es gibt im Diagramm nur dort auftauchen, wo sie gebraucht werden. Man muss im Büchiautomat also jedes mal alle hin schreiben. Ansonsten ist LSC *permissive*. Wenn eine hot Condition nich erfüllt ist, bleibe so lange im derzeitigen Zustand, bis sie erfüllt ist.
Regeln für *Fired Sets*:

① ② ③
$C \cap \mathcal{F} = \emptyset$ – $C \cup \mathcal{F}$ is a cut – only direct $\prec$-successors – same instance line on front pairwise unordered – sending of asynchronous reception already in
④

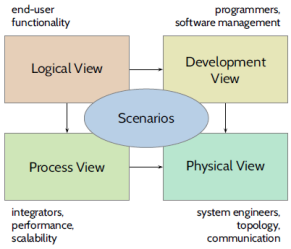




Legal exit: wenn cold conditions oder LocInvariants nicht erfüllt werden. *trivially satisfied:* bis zum Ende normal durchgehen. *Non-trivially satisfied*: bis zu einem akzeptierenden Zustand mittendrin gehen und nicht weiter, also ab da immer das gleiche senden. *Illegal exit(violates LSC):* hot condition/LovInv. Mit AC anfangen!!!!

The need to know principle: Jedes Interface eines Moduls gibt so wenig wie möglich preis darüber, wie es innendrin arbeitet.

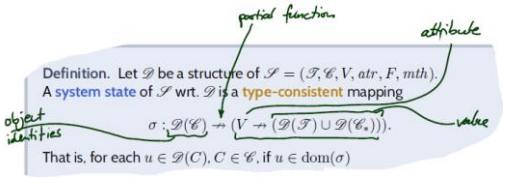
The 4 + 1 view:



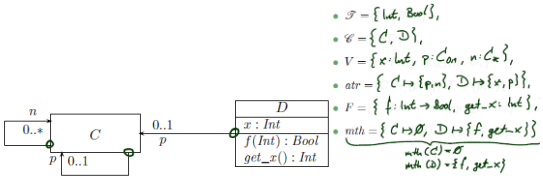
Structure vs Behavior:

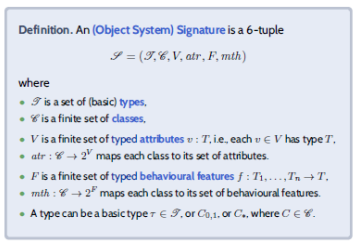
Object & Class diagram:

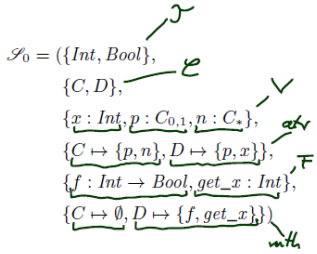
System state:

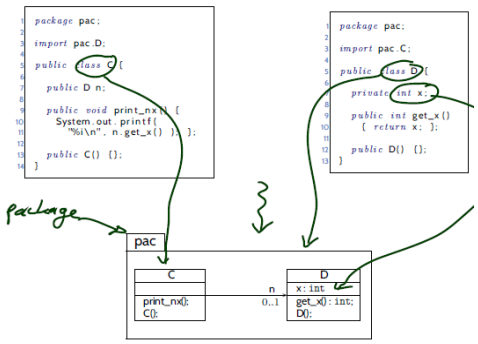


Class diagramm:

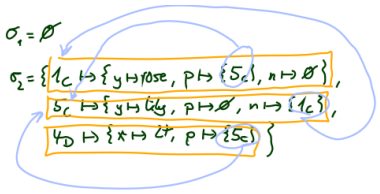








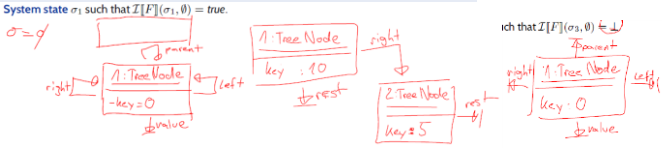
System state:
werte sind zugewiesen -> also kein class Diagramm mehr sondern system state.



OCL:

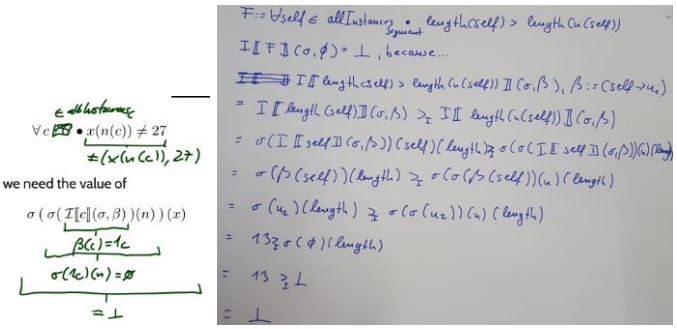
$$F := \forall\, self \in allInstances_{TreeNode} \bullet key(self) \geq key(leftChild(self)) \wedge key(self) \leq key(rightChild(self))$$

System state σ_3 such that $\mathcal{I}[\![F]\!](\sigma_3, \emptyset) = true$.



false > ⊥ > true

- Recall **prefix notation**: $\forall c \in allInstances_C \bullet \neq(x(c), 27)$
  Note: $\neq$ is a binary function symbol, 27 is a 0-ary function symbol.
- **Example**:
  $\mathcal{I}[\![\forall c \in allInstances_C \bullet \neq(x(c), 27)]\!](\sigma, \emptyset) = true$, because...
  $\mathcal{I}[\![\neq(x(c), 27)]\!](\sigma, \beta)$, $\beta := \emptyset[c := 1_C] = \{c \mapsto 1_C\}$
  $= \neq_{\mathcal{I}}(\mathcal{I}[\![x(c)]\!](\sigma, \beta), \mathcal{I}[\![27]\!](\sigma, \beta))$
  $= \neq_{\mathcal{I}}(\sigma(\mathcal{I}[\![c]\!](\sigma, \beta))(x), 27_{\mathcal{I}})$
  $= \neq_{\mathcal{I}}(\sigma(\beta(c))(x), 27_{\mathcal{I}})$
  $= \neq_{\mathcal{I}}(\sigma(1_C)(x), 27_{\mathcal{I}})$
  $= \neq_{\mathcal{I}}(13, 27) = true$ ...and 1_C is the only C-object in σ: $\mathcal{I}[\![allInstances_C]\!](\sigma, \emptyset) = \{1_C\}$.



Communicating finite automata:

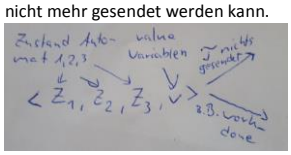
$\mathcal{N} \models E\langle\rangle$: Es gibt einen Pfad, auf dem ein Knoten folgende Eigenschaft hat.

$\mathcal{N} \models E[]$: Es gibt einen Pfad, auf dem alle Knoten folgende Eigenschaften haben.

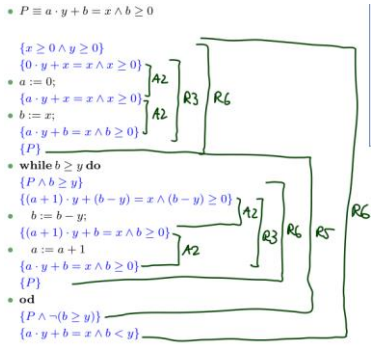
$\mathcal{N} \models A\langle\rangle$: Auf allen Pfaden gibt es einen Knoten, der folgende E. hat.

$\mathcal{N} \models A[]$ WorkerA.error $\vee$ not deadlock: Auf allen Pfaden haben alle Knoten die Eigenschaft.

<u>*Deadlock*</u>: Wenn die Automaten in einem Zustand festhängen, wenn z.B. etwas benötigtes nicht mehr gesendet werden kann.



Calculus PD:



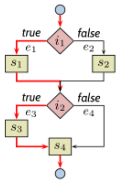
Software Testing:

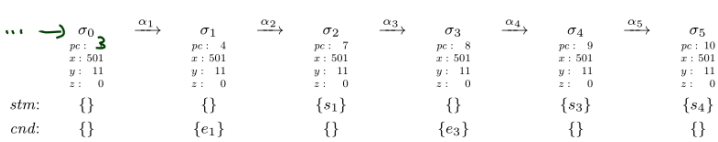
<u>*control flow graph:*</u>

```
1: int f(int x, int y, int z)
2: {
3:   i1: if (x > 100 ∧ y > 10)
4:   s1:   z = z * 2;
5:       else
6:   s2:   z = z/2;
7:   i2: if (x > 500 ∨ y > 50)
8:   s3:   z = z * 5;
9:   s4: return z;
10:}
```

$Stm_f = \{s_1, s_2, s_3, s_4\}$

$Cnd_f = \{e_1, e_2, e_3, e_4\}$





In											test suite coverage		
x, y, z	i_1/t	i_1/f	s_1	s_2	i_2/t	i_2/f	c_1	c_2	s_3	s_4	% stm	% cnd	i_2/% term
$501, 11, 0$	✔		✔		✔		✔		✔	✔	75	50	25
$501, 0, 0$		✔		✔	✔		✔		✔	✔	100	75	25
$0, 0, 0$		✔		✔		✔				✔	100	100	75
$0, 51, 0$		✔		✔	✔			✔		✔	100	100	100

Soll/output nicht vergessen!!!!

s_k = statements; i_k = condition/branch coverage; term = % bei c_1 v c_2

<u>*useless statements:*</u> statements, die nie erreicht warden können, da bedingung nie true werden kann

C_1	C_2	Term %	Nur bei veroderung!
0	**0**	50	**False effective**
0	**1**	75	**True effective**
1	0	100	**True effective**

Ein Test ist erfolgreich (successful), wenn er einen Fehler findet. Er ist nicht erfolgreich, wenn er durch läuft und keinen Fehler findet.