Begriffliches

Organisation → beschreibt das geordnete arbeitsteilige Zusammenwirken mehrerer Komponenten eines Systems bei der Durchfuehrung zielgerichteter Handlungen

wesentlich sind → Disziplinen | Mechanismen des geordneten Zusammenwirkens

abstrakte Beschreibung durch
→ Agenten | Aktoren (aktive Komponenten) ⇝ fuehren Transaktionen | Operationen auf (Darstellungen von) Nachrichten | Dokumenten | Objekten einer realen | virtuellen Welt aus
→ Kanaele (passive Komponenten) ⇝ dienen als Traeger von Nachrichten | Objekten ... beim Transport zwischen Komponenten

System → ist Menge aktiver | passiver Komponenten, die zweckgebunden miteinander kooperieren | kommunizieren
→ wird beschrieben durch
⇝ Eigenschaften der Komponenten und deren Wechselwirkungen
⇝ Festlegung von Systemgrenzen
⇝ Beziehungen zwischen Ein-/ Ausgabe

Rechnerorganisation betrifft

$\rightarrow$ Organisation von algorithmisch | deklarativ spezifizierten Berechnungs- | Entscheidungsvorgaengen | Prozessen
$\rightarrow$ Programmausfuehrendes System

Aktoren $\rightsquigarrow$ sind vom System unterstuetzte Operatoren
Kanaele $\rightsquigarrow$ sind Speicherbereiche | Laufzeitumgebungen zur Kommunikation von Operanden zwischen Operatoren

$\rightarrow$ Ablaufplanung | Ressourcenverwaltung fuer das programmausfuehrende System | Abwickelung von Kommunikationsvorgaengen | Verwaltung grosser Datenbestaende
$\rightarrow$ Betriebssystem

Aktoren $\rightsquigarrow$ sind primaere | sekundaere Scheduler fuer Prozesse | Transaktionen | ...
Kanaele $\rightsquigarrow$ sind diverse Datenstrukturen zur Darstellung von Prozess | Ressource | Systemzustaenden

Rechner(System)Architektur betrifft

→ Gestaltung | Festlegung des Erscheinungsbildes | der Schnittstellen von Rechnersystemen gegenueber dem system-nahen Benutzer ⇝ System(kern)programmierer | Implementierer von Compiler-backends | Code-Generatoren

Architektur
→ des programmausfuehrenden Systems ist definiert durch
⇝ nicht-privilegierten Instruktionssatz | Adressierungsmodes | Elementardatentypen (Formate)
⇝ direkt zugaengliche Ressourcen (Funktionseinheiten | Register | Speicherbereiche ...)

→ des Betriebssystems ist definiert durch
⇝ privilegierten Instruktionssatz (Prozessmanagement | i/o Vorgaenge | Prozess-Kommunikation)
⇝ Unterbrechungssystem | Prozesswechselmechanismen | ...

Architekturen definieren Systemfamilien mit identischer Funktionalitaet ⇝ Maschinenprogramme werden auf allen Modellen der Familie auf gleiche Weise ausgefuehrt | produzieren exakt gleiche Ergebnisse

Essentials of system architecting

$\rightarrow$ bringing form to functionality | purpose
$\rightarrow$ fitting the interfaces (among components | to the user)
$\rightarrow$ reducing complexity
$\rightarrow$ balancing the needs
$\rightarrow$ compromizing among the extremes

$\rightarrow$ consistency (absence of contradictory behavior)
$\rightarrow$ no unnecessary functions (propriety)
$\rightarrow$ no functional reduncies in different diguises (parsimony)
$\rightarrow$ transparency
$\rightarrow$ generality and completeness of functions

$\rightarrow$ aiming for optimality $\leadsto$ a waste of time

$\rightarrow$ primary objective of architecting
   $\rightarrow$ KISS $\leadsto$ keep it simple, stupid !!!!

$\rightarrow$ well designed systems with good chances of long-term survival are the product of only a few !!!!

Computer Engineering

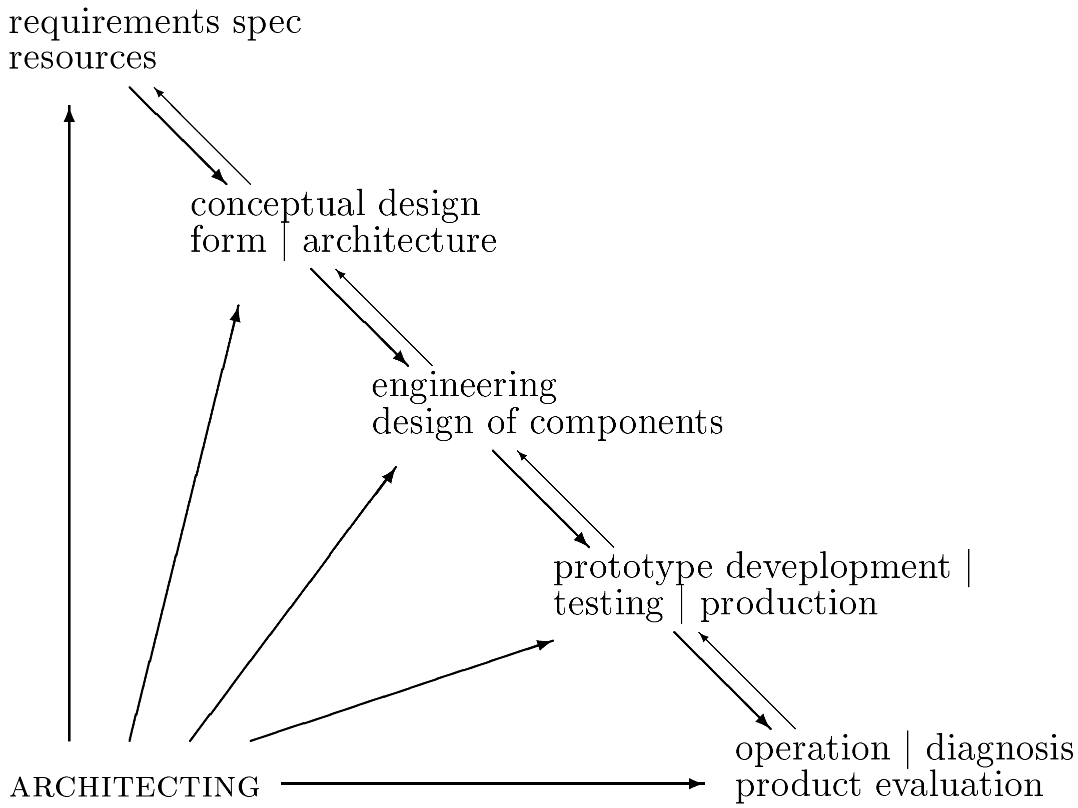
$\rightarrow$ technische Realisierung von individuellen Modellen einer Architektur | Systemfamilie

Wesentliche Gesichtspunkte sind
$\leadsto$ technologische Gegebenheiten
$\leadsto$ Produktions | Vertriebs | Wartungskosten
$\leadsto$ Leistungs | Kapazitaets | Zuverlaessigkeitsanforderungen
$\leadsto$ Details der Soft | Firm | Hardware Implementierung (wird ggf. der Architektur zugerechnet | nicht genau zu trennen)

Wichtig :

$\rightarrow$ das Architecting beeinflusst alle Phasen der Systementwicklung | Fertigung | Nutzung
$\rightarrow$ idR muessen die einzelnen Phasen der Systementwicklung mehrfach durchlaufen werden, da sie sich wechselseitig beeinflussen

$\leadsto$ Wasserfall Modell



Instruction set mnemonics

class	mnemonic	operation
move instr	MOVE.DU	moves unpacked decimals mem_to_mem
	MOVE.C	moves character strings mem_to_mem
	PK.DU	packs decimals mem_to_mem
	UNPK.DP	unpacks packed decimals mem_to_mem
	MOVE.X	moves fixed formats reg_to_reg
	LD.X	loads fixed formats mem_to_reg
	ST.X	stores fixed formats reg_to_mem
	LDM.X	loads multiple fixed formats mem_to_reg
	STM.X	stores multiple fixed formats reg_to_mem
	LDA.W	loads address into reg
	SHL.U	shifts fixed formats left
	SHR.U	shifts fixed formats right
	CLR.X	clears fixed formats (registers)
arithmetic/ logic instr	ADD.X	addition
	SUB.X	subtraction
	MULT.X	multiplication
	DIV.X	division
	INC.U	increment
	DEC.U	decrement
	COMP.U	compares integers
	CMP.C	compares character strings
	CMP.DP	compares packed decimals
branch instr	BC.CC	(un-)conditional branch
	BR_DEC	branch on decrement (loop control)
	BSR	branch to subroutine
	RET	return from subroutine

Meaning of the subscripts

<table>
<tr><th>modifier</th><th>type | format</th></tr>
<tr><td>B</td><td>integer byte | 8 bits</td></tr>
<tr><td>H</td><td>integer half word | 16 bits</td></tr>
<tr><td>W</td><td>integer (full) word | 16 bits</td></tr>
<tr><td>F</td><td>floating point full word | 16 bits</td></tr>
<tr><td>D</td><td>floating point double word | 64 bits</td></tr>
<tr><td>C</td><td>character string | ≤ 256 bytes</td></tr>
<tr><td>DD</td><td>decimal string unpacked | ≤ 16 bytes (digits)</td></tr>
<tr><td>P</td><td>decimal string packed | ≤ 16 bytes (digits)</td></tr>
<tr><td>X</td><td>B | H | W | F | D</td></tr>
<tr><td>U</td><td>B | H | W</td></tr>
</table>

Branch condition codes

CC mnemonic	meaning
`T`	*true* → branch always
`F`	*false* → branch never (go in-line)
`V`	arithmetic result overflow
`C`	arithmetic carry out
`N`	negative arithmetic result
`Z`	zero arithmetic result

Note that condition codes are
→ set by arithmetic | logic instructions;
→ inspected by subsequent `BC.CC` instructions in which `CC` may either be specified as `T` or `F` or as any combination of `V | C | N | Z`. as condition code.