KIV/ZSWI 2003/2004
Přednáška 8

Strukturovaná analýza a návrh systému

* strukturované metodiky pro analýzu a návrh systému historicky předcházely objektovým metodikám
 - objektově-orientovaný vývoj - data a nad nimi pracující funkce chápeme jako objekty
 - strukturované - na funkce a na data se zaměřují víceméně odděleně
  . odpovídá strukturovanému programování
  . funkce jsou aktivní a mají chování
  . data jsou pasivní, ovlivněna funkcemi
  . fce systému postupně rozdělujeme shora dolů na části, nejčastěji pomocí diagramů datových toků (data-flow diagrams)

* dnes se všeobecně dává přednost OO metodikám, ale strukturované metodiky mohou stále ještě posloužit v následujících případech:
 - malé programy (několik set řádek kódu): příliš jednoduché, aby se vyplatilo vytvářet třídy
 - programy s krátkou dobou života, např. prototypy, které budou zahozeny (pokud cílem není získat představu o vytvářených třídách): opět se nemusí vyplácet vytvářet třídy
 - pokud se pravděpodobně bude měnit funkčnost, ale ne data: v OO přístupu by funkčnost byla rozprostřena mezi více objektů, proto může být výhodnější strukturovaný přístup (pokud se naopak budou měnit data, je OO přístup výhodnější, protože změny jsou zapouzdřeny do jednotlivých objektů)

* výhoda strukturovaných metodik
 - metodiky mohou být jednodušší, vytvářené modely mohou být srozumitelné zákazníkovi => zákazník se snáze účastní strukturované analýzy
 - návrh systému může být i rychlejší (nevytváříme přídavnou strukturu tříd)
* nevýhody strukturovaných metodik
 - jsou považovány za nemoderní
 - výsledný systém se většinou hůře udržuje

* podobně jako u objektové analýzy a návrhu se vytvářejí modely = abstrakce klíčových vlastností studovaného systému
 - model slouží jako vstup do dalších fází SW procesu:
  . model kontextu systému - určuje hranice vytvářeného systému
  . modely strukturované analýzy: diagramy datových toků, datový slovník, ERA diagramy, specifikace činností procesů
  . modely strukturovaného návrhu: strukturogramy (structure charts) - na jejich základě můžeme zahájit kódování
 - vytváření modelu může mít podporu CASE nástrojů (editor modelů, částečná kontrola modelu, automatická tvorba dokumentace)

Poznámka pro zajímavost (CASE nástroje podporující strukturované metodiky)

Srovnání některých CASE nástrojů podporujících strukturované metodiky můžete najít např. v článku V. Řepy: Programování ve velkém, Softwarové noviny 5/2003.

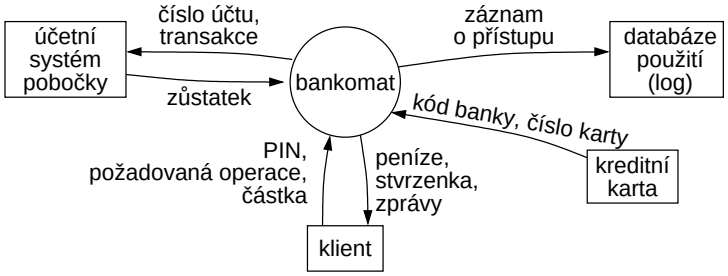
[]

Model kontextu systému

* již úplně na začátku získávání požadavků je třeba určit hranice systému, tj. co bude tvořit systém a co bude okolí systému
* v některých případech nemusí být úplně zřejmé: např. evidence příjmu dřeva v papírně = váha, terminály pro vstup informací o objemu a kvalitě dřeva, databáze...
* zvolenou hranici často určují netechnické faktory, např. jí určíme tak, abychom měli co nejvíce věcí pod kontrolou

* po definici hranic určíme kontext a závislosti systému na okolí

* obvykle znázorňujeme nakreslením jednoduchého diagramu kontextu (context diagram)
 - hranice vytvářeného systému je znázorněna kolečkem s vepsaným názvem systému
 - lidé, organizace nebo jiné systémy se kterými náš systém komunikuje se znázorňují pojmenovaným obdélníkem; ve strukturované analýze se nazývají terminátory (mají stejný význam jako aktéři v OO analýze)
 - vstupující a vystupující data jsou znázorněna šipkami mezi systémem a terminátorem
 - například model kontextu systému pro bankomat:

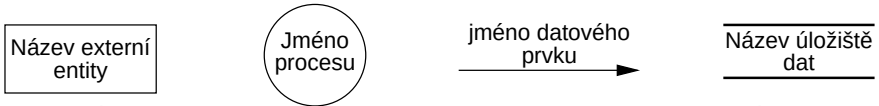


Diagramy datových toků
---

* angl. data-flow diagram, DFD, ale někdy také data-flow graph, work flow diagram, function model, bubble chart, process model...
* součástí různých strukturovaných metod cca 1955-1990 (ale i některých OO metodik, např. OMT), v literatuře užívány různé notace
* DFD jsou jednoduché a intuitivní (je možné je vysvětlit zákazníkovi)
* lze je použít pro modelování toku dat SW systémem, ale také modelování toku dat a fyzických předmětů (materiálu) v organizaci

* my budeme především modelovat tok dat SW systémem
* data jsou zpracovávána posloupností kroků (kroky provádějí lidé, funkce programu)

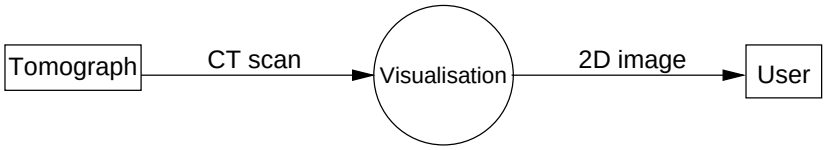
* základní notace pro DFD:
 - externí entita neboli terminátor
  . producent nebo konzument dat (začíná nebo končí v něm tok dat)
  . je mimo hranice modelovaného systému
  . může být osoba, jiný systém, hardware apod.
  . znázorněna obdélníkem s názvem uvnitř
 - proces
  . provádí transformaci dat
  . znázorněn kolečkem (bublinou), je vhodné bubliny číslovat a nutné konkrétně pojmenovat - sloveso + podstatné jméno (”ověř telefonní číslo”)
 - tok dat
  . reprezentuje ”data v pohybu”
  . znázorněn šipkou, šipka ukazuje směr toku dat; datový prvek má být pojmenován
 - paměť (datový sklad)
  . úložiště dat pro použití jedním nebo více procesy, pracujícími v různých časových obdobích
  . znázorněn dvojitou čarou
  . pokud je vyjímána pouze jedna datová položka, nemusíme označovat datový tok



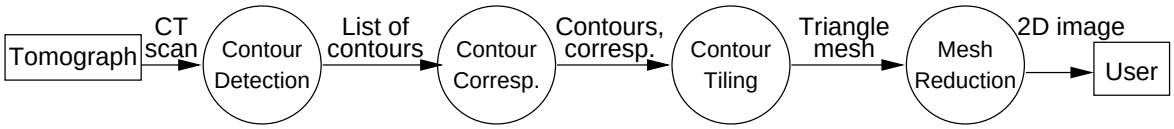
a) externí entita *b) proces* *c) tok dat* *d) úložiště dat*

* DFD může být použit pro reprezentaci systému libovolné úrovně abstrakce
 - začínáme na DFD úrovně 0 = fundamentální model systému, je vlastně totéž co

již popsaný model kontextu systému
- celý SW systém je zakreslen jako jedna bublina, má jeden nebo více vstupů a výstupů
- například vizualizace snímků z tomografu:
 . vstupem je množina 2D řezů tělem pacienta
 . výstupem je 2D pohled na snímek



- systém můžeme rozdělit do menších částí a znázornit na větší úrovni podrobnosti:



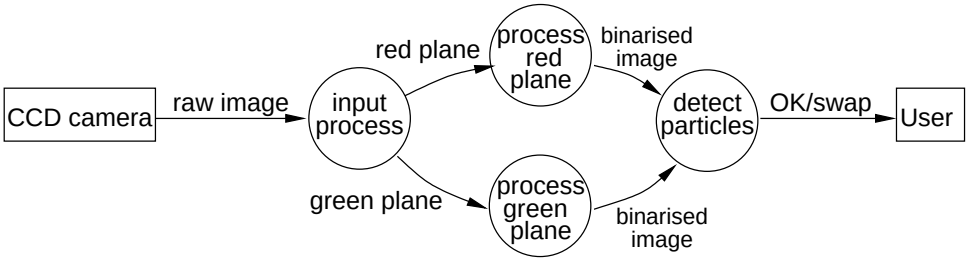
* výhodná vlastnost DFD - model může být postupně zjemňován

- např. fundamentální model systému "Monitor kvality oleje" má vstup "obrázek sejmutý CCD kamerou mikroskopu", výstupem je rozhodnutí zda je třeba olej vyměnit:

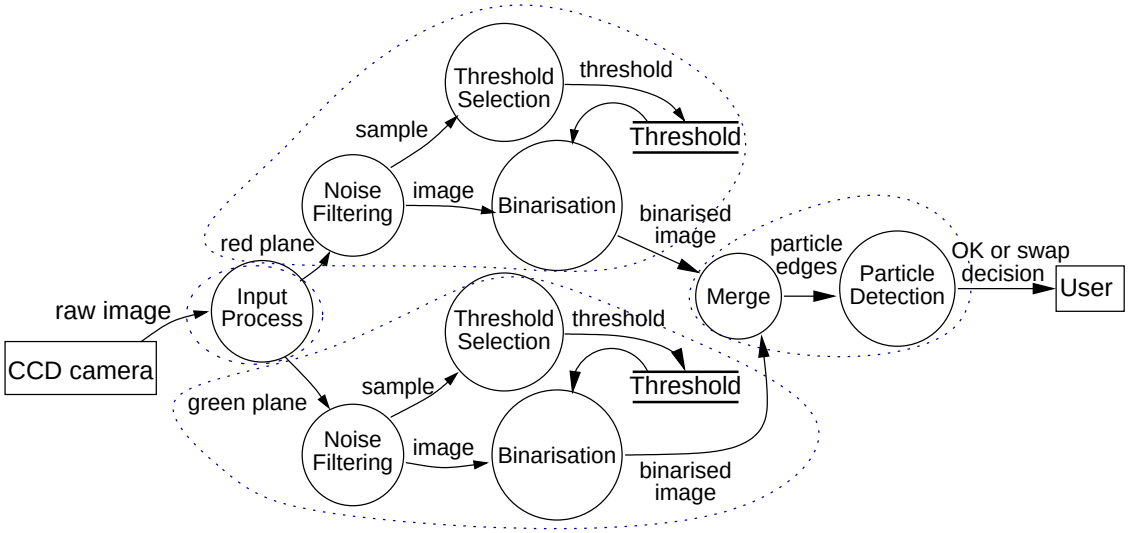
L0 DFD:

- zjemníme model systému
 . najdeme kandidáty procesů, datových toků a pamětí
 . všechny šipky, bubliny a úložiště smysluplně pojmenujeme
 . musí být udržena "kontinuita toku dat", tj. původní vstupy a výstupy musejí zůstat zachovány
 . pokud je DFD nepřehledný, překreslíme ho
- např. výše uvedený systém rozdělíme do čtyř transformací:

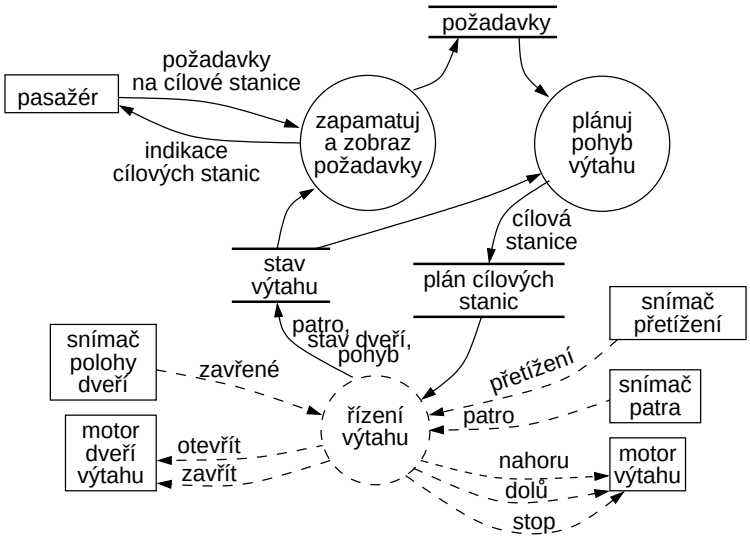
L1 DFD:



- v dalším kroku postupujeme ve zjemňování bublinu po bublině (tečkováním jsem naznačil obsah bublin z L1 DFD):




```
 - vytvořené DFD by neměly být příliš velké - měly by se bez problémů vejít
   papír velikosti A4

Co zkontrolovat:

* černé díry - bubliny které mají vstup ale nemají výstup
* spontánní generátory - mají výstup ale žádný vstup; jediný rozumný případ
  tohoto typu je generátor náhodných čísel
* neoznačené procesy a datové toky - častým důvodem pro výskyt je to, že
  analytik není schopen najít pro ně rozumný název, např. protože sdružují
  nesouvisející procesy/toky dat (pak je řešením proces nebo datový tok
  rozdělit)

Poznámka (rozšíření DFD)

 Výše uvedená základní varianta DFD je vhodná pro modelování systémů které
 jsou řízeny datovými vstupy a kde se nezpracovávají téměř žádné vnější
 události. Zcela jinak je tomu v RT systémech; pro taková použití byla
 navržena celá řada rozšíření základního DFD. Tok řízení se většinou
 znázorňuje přerušovanou šipkou a řídící proces jako bublina zakreslená
 přerušovanou čarou. Například v následujícím DFD je řídícím procesem ”řízení
 výtahu”:
```




```
[]

* v dalším kroku potřebujeme
 - vytvořit definici dat
 - specifikovat logiku procesů
  . na konečné úrovni zjemnění doplňujeme popis specifikací procesu
    buď v přirozeném jazyce nebo v pseudokódu
```

Datový slovník

* v DFD
 - šipky reprezentují tok dat
 - paměť je množina datových položek
 - je třeba mít metodu reprezentace obsahu dat v datových tocích a pamětech
 - pro popis se používá datový slovník = seznam datových prvků s definicemi

Důležitost popisu dat viz ”rozhovor s marťanem” (Yourdon 1989):

M: Co je to vlastně jméno?
Z: To je jak se navzájem nazýváme.
M: (zmateně): Znamená to, že se nazýváte jinak, když jste šťastní než když
   máte vztek?
Z: (trochu překvapeně, že M je snad mimozemšťan): Ne, jméno je pořád stejné.
M: (konečně pochopil): Aha, rozumím, u nás máme to samé. Moje jméno je 3.1415.
Z: Ale to je přece číslo, ne jméno.
atd.; v praxi často zákazník považuje analytika za marťana.

Např. při určení ”co je to jméno” se můžete setkat s problémy:

* musí mít každý křestní jméno a příjmení? Co je ”křestní jméno” a co
  je ”příjmení” např. pro jméno ”Ing. Tran Quoc Trung”?
* jaké znaky mohou být součástí jména? Co ”Kropáčková-Jouzová” nebo ”D’Arcy”?
* jak budeme zacházet a dodatky typu ”Jiří Stivín III.”?

* proto vytváříme formálnější definici datového slovníku
* popisovaná data jednoduchá nebo složená
 - jednoduchá - známé typy dat, je třeba zadat obor hodnot, příp.
   použité jednotky, příp. přesnost
 - složená - popíšeme odkazem na jednoduché položky

Notace podle Yourdona (1989):

* neterminální symboly uvádím malými písmeny

symbol	význam	příklad	vysvětlení
=	skládá se z	X = Y	X se skládá z Y
+	a	Z = X + Y	Z se skládá z X a Y
()	může být vynecháno	Z = X + (Y)	Z se skládá z X a případně Y
{}	opakování	Z = { X }	Z se skládá z libovolného počtu X
	- pokud je nutné určit dolní resp. horní mez počtu opakování, zapisuje se před resp. za složené závorky:		
		Z = 1{X}	Z se skládá z jednoho nebo více X
		Z = {X}1	Z se skládá z 0 nebo jednoho X
		Z = 1{X}10	Z se skládá z 1 nebo 2 nebo ... 10 X
[]	jedna z možností	Z = [X | Y]	Z se skládá buď z X nebo z Y
**	komentář	*toto je komentář*	
@	klíčová položka		
@<číslo>	část složeného klíče		

Jméno by mohlo být definováno následovně:

```
jméno = (tituly) + @<2>křestní_jméno + (@<3>prostřední_jméno) +
                   @<1>příjmení + (vědecká_hodnost)
tituly = {titul}
titul  = [ Pan | Paní | Dr. | Ing. | RNDr. | MUDr. | JUDr. | Prof. | Doc. ]
vědecká_hodnost = [ CSc. | DrSc. ]
křestní_jméno    = platné_jméno
prostřední_jméno = platné_jméno
příjmení         = platné_jméno
platné_jméno = velké_písmeno + { písmeno }
písmeno = [ velké_písmeno | malé_písmeno ]
velké_písmeno = *velká písmena české abecedy*
 [ A | Á | B | C | Č | ... | Z | Ž ]
malé_písmeno  = *malá písmena české abecedy*
 [ a | á | b | c | č | ... | z | ž ]
```

Vše co se nedá popsat výše uvedeným způsobem jako komentář:
 - význam datového prvku v kontextu aplikace (pokud je stejný jako
   název, pak se uvádí prázdný komentář ”**”)
 - z čeho se prvek skládá, pokud je složen ze smysluplných elementů
 - rozsah hodnot, pokud element nelze dále dekomponovat; případně přesnost
   jako počet významných číslic za desetinnou čárkou

Příklady:

```
      výška    = *výška pacienta při přijetí do nemocnice*
                 *jednotka: cm; rozsah: 10-250*

      hmotnost = *hmotnost pacienta při přijetí do nemocnice*
                 *jednotka: kg; rozsah: 1-200; přesnost: 0.1 kg*

      datum-narození = **
                 *jednotka: počet dní od 1.1.1900; rozsah: 0-73000*
```

[]

* každá šipka v DFD by měla mít popis v datovém slovníku

* ve velkých systémech může mít datový slovník několik tisíc položek
* pro definici, kontrolu konzistence a úplnosti je vhodné používat
  nástroje, jsou součástí DB systémů, notace se může poněkud lišit

* je velmi žádoucí udržet co nejmenší míru redundance kvůli lokalizaci změn
* pro případná synonyma vždy pouze jednu definici

```
      zákazník = jméno + adresa + kategorie
      klient   = zákazník
                 *synonymum pro ”zákazník”*
```

* kdybychom uvedli dvě (stejné) definice, je nebezpečí, že při změně jednu z
  nich zapomeneme opravit
* někteří autoři doporučují zvýraznit to, že se jedná o synonymum, uvedením
  hvězdičky ve všech výskytech (např. klient*), definici bychom zapsali
  takto:

```
         klient* = zákazník
```

ERA diagramy
----

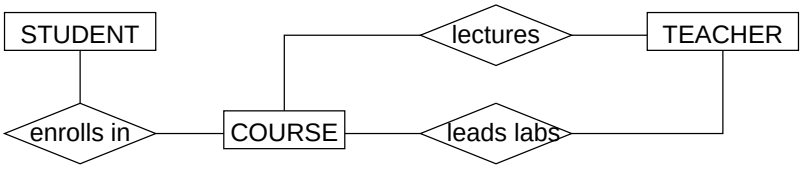
* v češtině také entitně-relační diagramy, E-R diagramy, v angl.
  nejčastěji entity-relationship diagrams (E-R diagrams, ERD)
* popisují strukturu uchovávaných dat na vysoké úrovni abstrakce
* základy Chen 1976, později vývoj různými směry - existuje mnoho notací,
  zde uvedeme pouze základní notaci

Poznámka pro zajímavost (ERA diagramy a diagram tříd v UML)

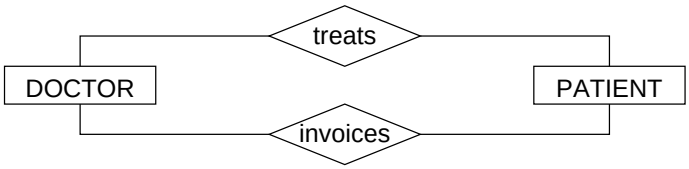
 Notace pro diagramy tříd v UML se historicky vyvinula z notace pro ERA
 diagramy, tj. podobnost mezi nimi není náhodná.

[]

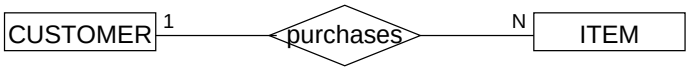
* při vytváření systému si klademe otázky:
 - které údaje musíme uchovávat v systému?
 - jaký je vzájemný vztah údajů?
* zakreslujeme nejčastěji pomocí ERA diagramu, např.:
 - student volí předměty, předměty učí a cvičí učitelé
 - přehledně můžeme nakreslit následovně:



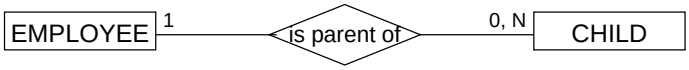
kde:
* obdélníkem zakreslujeme typy objektů (datové entity)
 - představuje množinu navzájem rozlišitelných objektů reálného světa
* kosočtvercem zakreslujeme množinu vztahů (relace)
 - relace = množina vztahů které si systém musí zapamatovat (nelze je odvodit)
 - mezi dvěma entitami může být více relací



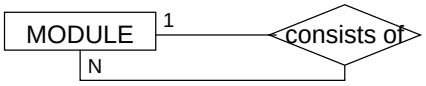
 - relace charakterizována aritou = počet objektů účastnících se vztahu
 - nejčastěji se rozlišuje 1:1, 1:N, M:N



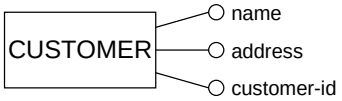
 - nepovinný (volitelný) vztah = nemusí nastat



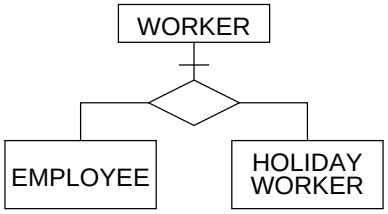
 - můžeme popsat i rekurzivní vztah



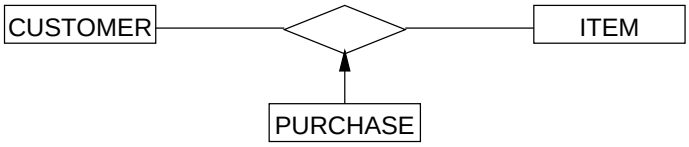
* atributy
 - popisují entity; entita popsána pomocí jednoho nebo více atributů
 - atributy se týkají všech instancí entity
 - v ERA diagramu se někdy znázorňují malým pojmenovaným kolečkem



* někdy můžeme zjistit, že různé entity jsou ve skutečnosti pouze odlišné
  formy základní entity
 - například všichni zaměstnanci mají jméno, adresu, rodné číslo
 - stálí zaměstnanci mají navíc měsíční plat a osobní ohodnocení
 - brigádníci mají navíc hodinovou mzdu
 - jinými slovy nadtyp je popsán atributy, které se týkají všech podtypů
 - analytik může znázornit pomocí nepojmenovaného kosočtverce s přeškrtnutím
   vztahu k nadtypu:



* někdy zjistíme, že potřebujeme uchovávat také informaci o relaci
 - příklad: zákazník nakoupil zboží
  . relace ”nakoupil” sdružuje zákazníka a jednu nebo více položek zboží
  . pokud bychom chtěli uchovat informaci o datu nákupu, tato informace
    nepatří ani k zákazníkovi, ani ke zboží
  . zavedeme ”nakoupil”, má fci typu objektu (tj. má atributy) a zároveň
    vztahu (spojuje zákazníka s položkami zboží)
 - nazývá se asociativní indikátor typu objektu nebo průnikový typ
 - znázorňuje se následovně:



* ERA diagramy vstup pro návrh databáze, např. ”pamětí” v DFD
* zde byly uvedeny pouze základy notace, podrobnější popis viz např.:

  Jaroslav Pokorný: Konstrukce databázových systémů.
  Vydavatelství ČVUT 1999

Specifikace činnosti procesů

* zjemňováním DFD nám vznikl rozklad problému na elementární procesy
  komunikující pomocí datových toků
* nyní potřebujeme specifikovat, co se děje v elementárním procesu

* pro specifikace procesů se používají následující nástroje:
 - pseudokódy nebo jejich grafický ekvivalent
 - rozhodovací tabulky
 - rozhodovací stromy
 - konečný automat (můžeme použít konvenci stavového diagramu z UML)

Pseudokódy

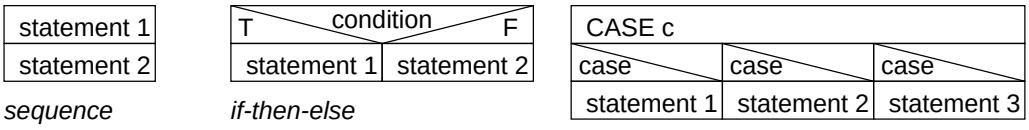
* jako příklad uvedu Yourdonovu ”strukturovanou angličtinu”, resp.
  ”strukturovanou angločeštinu”

* základní myšlenka - omezený slovník běžného jazyka, přidáme strukturu
  programovacího jazyka
* slovník se skládá z:
 - příkazů
 - rozhodovacích konstrukcí
 - opakovacích konstrukcí

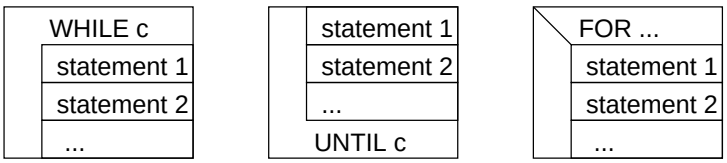
* příkazy:
 - výraz, např. X = Y*Z
 - sloveso, pojmy definované v datovém slovníku
  . pro jakoukoli specifikaci by mělo postačovat 40-50 sloves, např.
    READ, WRITE, CREATE, APPEND, FIND, ... resp. české načti, vypiš, vytvoř,
    přidej atd.
 - vnoření konstrukcí nejčastěji vyjádřeno odsazením, někdy se můžeme setkat
   také s číslováním 1.1.1, 1.1.1.1...
* rozhodovací konstrukce:

```
IF podmínka            IF podmínka            DO CASE
    akce                   akce1              CASE podmínka1
ENDIF                  ELSE                       akce1
                           akce2                  ...
                       ENDIF                  CASE podmínka_n
                                                  akce_n
                                              OTHERWISE
                                                  akce
                                              ENDCASE
```

```
* cykly:
  DO WHILE podmínka      REPEAT
      akce                   akce
  ENDDO                  UNTIL podmínka

Příklad:

  PROCES 1.2.3: Účtování objednávek.
  DO WHILE jsou objednávky ke zpracování
      READ objednávka
      celková-částka = 0
      DO WHILE jsou položky v objednávce
          celková-částka = celková-částka + cena-položky
      ENDDO
      WRITE (do procesu 1.1.5) ID-zákazníka + celková-částka
  ENDDO


* definice procesu by neměla přesáhnout jednu stránku A4 (cca 70 řádek)
 - pokud se nevejde, měla by se použít jiná formulace, resp. jednodušší
   algoritmus
 - pokud to nejde, možná že proces není elementární


Nassi-Shneidermanovy diagramy
.............................

* někdy se používají místo pseudokódu; uvádím pro zajímavost
```

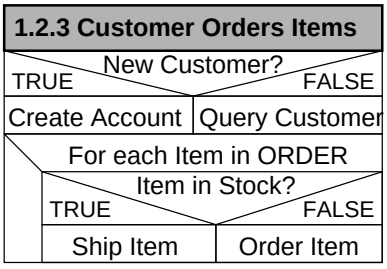


sequence *if-then-else*



while-do *repeat-until* *for-do*

```
* příklad Nassi-Shneidermanova diagramu procesu:
```

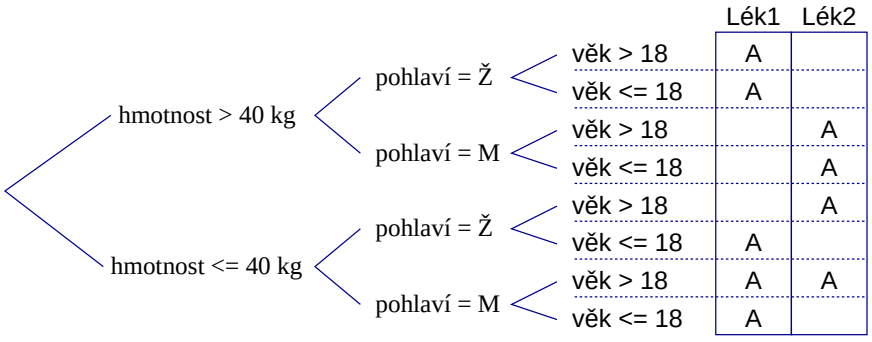



```
* po vysvětlení jsou pro zákazníky často srozumitelnější než pseudokód (podle
  publikovaných výzkumů pro 75-80% lidí)
* vyžaduje nemalé množství grafiky - vyplatí se pouze pokud máme SW podporu
  pro jejich vytváření

* neměli bychom překročit 3 úrovně vnoření, jinak je vhodná spíše rozhodovací
  tabulka

Vstupní a výstupní podmínky
...........................

* v některých případech se udává pouze vstupní a k ní odpovídající výstupní
```

```
  podmínka:

  PROCES 1.2.3: Účtování objednávek.
  Vstupní podmínka:
      na vstupu (z procesu 1.1.4) se objeví objednávka
  Výstupní podmínka:
      na výstup (do procesu 1.1.5) je zasláno ID-zákazníka + celková-částka
  Vstupní podmínka:
      na vstupu (z procesu 1.1.4) je objednávka, zákazník není v databázi
  Výstupní podmínka:
      je vygenerována chybová zpráva

* běh procesu je spouštěn určeným vstupem

Rozhodovací tabulky a stromy
............................

* používají se pokud by byl pseudokód vyjadřující podmínku složitý

* nejprve najdeme všechny relevantní vstupy, učíme počet možných kombinací
* vytvoříme tabulku
 - sloupce = pravidla: počet pravidel je dán počtem kombinací vstupů
 - řádky tabulky: nejprve všechny vstupy, pak všechny výstupy
```

	1	2	3	4	5	6	7	8
věk > 18	A	N	A	N	A	N	A	N
pohlaví = Ž	A	A	N	N	A	A	N	N
hmotnost > 40 kg	A	A	A	A	N	N	N	N
podávat lék 1	A	A				A	A	A
podávat lék 2			A	A	A		A	

```
* horní polovina tabulky definuje podmínky
* ve spodní polovině je pro každý sloupec (= podmínku) uveden výstup

* pokud uživatel se kterým musíme konzultovat odmítne tabulku jako
  nesrozumitelnou, můžeme použít rozhodovací strom:
```




```
* výhoda obou způsobů:
 - specifikací není určen způsob implementace
 - s uživatelem můžeme probrat jedno pravidlo po druhém
 - a hlavně máme jasnou odpověď pro všechny kombinace podmínek

Strukturovaná analýza
---------------------

V předcházejícím textu jsme popsali nástroje potřebné pro strukturovanou
analýzu, ale zatím jsme neuvedli, jak tyto nástroje použít.

V dalším textu si proto uvedeme základy dvou nejznámějších metodik, a to
klasické DeMarcovy metodologie a Yourdonovy "moderní strukturované analýzy".

* vstupem strukturované analýzy (DeMarco) jsou uživatelské požadavky,
```

výstupem je tzv. strukturovaná specifikace
- systém je specifikován pomocí DFD, uvedeny podstatné procesy, paměti a údaje
- případné jednodušší procesy v DFD nižší úrovně
- elementární procesy zapsány v pseudokódu, rozhodovací tabulkou nebo stromem
- v datovém slovníku popis dat

Otázka: co má analytik vyrobit - model původního systému nebo nového?

* předpokládejme:
 - existující systém s ne zcela zřetelnou strukturou plní určité funkce
 - systém z nějakého důvodu nahrazujeme novým systémem
 - jak najít specifikaci nového systému?

* klasická strukturovaná analýza pomocí vytvoření 4 modelů systému:
1. fyzický model stávajícího systému (jaký systém používá zákazník?)
   - analytik zmapuje stávající systém, jeho fční strukturu a data
   - může obsahovat zpracovávaní a přesun fyzických formulářů apod.
2. logický model stávajícího systému (jaká je jeho logická struktura?)
   - z fyzického modelu vytvoříme logický (cca 75% redukce)
   - zrušíme všechny implementační detaily, co by systém dělal kdybychom měli ideální technologii (nekonečné paměti, nekonečnou rychlost atd.)
   - získáme logické procesy a podstatu transformace dat
3. logický model nového systému (co je třeba změnit?)
   - většina systému pravděpodobně zůstane stejná + požadavky na nové fce
   - po konzultaci s uživatelem promítnuty změny do logického modelu
4. fyzický model nového systému (jak to nejlépe implementovat?)
   - návrh implementace.

Problémy při striktním dodržování modelu:

* při vytváření fyzického modelu ví uživatel o systému víc než analytik; u uživatele tím často vznikne dojem, že analytik problematice nerozumí a že se jí teprve za jeho peníze učí
* uživatel odmítá spolupráci na vývoji nového logického modelu; má pocit, že pokud analytik neumí vytvořit bez pomoci zákazníka fyzický model stávajícího systému, pak nemůže umět ani dobře navrhnout nový systém
* analytickou práci někteří uživatelé považují za oddech vývojářů před ”skutečnou prací” (kódováním), tvorba 4 modelů tuto dobu prodlužuje, což snižuje ochotu spolupracovat s analytikem (tj. 4 modely je prostě moc)
* zmíněné problémy řeší ”moderní strukturovaná analýza” (Yourdon 1989)

Vyvažování modelů

Strukturovaná analýza:
* model kontextu systému
* diagramy datových toků (DFD)
* datový slovník
* ERA diagramy
* specifikace činnosti procesů
 - pseudokódy
 - Nassi-Shneidermanovy diagramy
 - rozhodovací tabulky a stromy

* každý model se zabývá nějakým aspektem modelovaného systému
 - DFD, specifikace procesů - funkce systému
 - datový slovník, ERA diagramy - data systému
* ve velkých systémech by bylo snadné vytvořit několik nekonzistentních pohledů na systém
 - např. v datovém slovníku mohou zůstat položky vzniklé v počáteční fázi analýzy systému, které se v DFD už nevyskytují
* proto je po dokončení modelů třeba provést tzv. vyvažování modelů

* vyvažování DFD a datového slovníku = zajistíme následující:
 - každý datový tok a každá paměť musí být definována v datovém slovníku
 - každá datová položka v datovém slovníku se musí vyskytovat v DFD
 - mechanická práce => je vhodné mít podporu CASE nástroje
* vyvažování DFD a specifikace procesů

- každá bublina v DFD musí být sdružena buď s DFD nižší úrovně nebo se specifikací procesu
- každá specifikace procesu musí mít bublinu v DFD nejnižší úrovně
- vstupy a výstupy si musejí vzájemně odpovídat

* vyvažování specifikace procesů a datového slovníku
- všechna data použitá ve specifikaci procesu musejí být definována buď lokálně nebo v datovém slovníku
- každá položka v datovém slovníku musí být odkazována ze specifikace procesu nebo z jiné položky datového slovníku

* vyvažování ERA diagramu oproti DFD a specifikaci procesu
- každá paměť v DFD musí odpovídat entitě, relaci nebo entitě+relaci v ERA diagramu
- položky v datovém slovníku popisují jak toky v DFD tak entity v ERA modelu
- procesy musejí být schopny:
  . vytvářet a rušit instance všech entit a relací v ERA diagramu
  . nastavovat hodnotu a používat hodnotu instance.

❇