Strukturovaná analýza a návrh systému

- strukturované metodiky pro analýzu a návrh systému historicky předcházely objektovým metodikám

- na funkce a na data se zaměřují víceméně odděleně
  - odpovídá strukturovanému programování
  - funkce jsou aktivní a mají chování
  - data jsou pasivní, ovlivněna funkcemi

- funkce systému postupně rozdělujeme shora dolů na části, nejčastěji pomocí diagramů datových toků (data-flow diagrams)

- dobře slouží 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: (pokud se naopak budou měnit data, je OO přístup výhodnější, protože změny jsou zapouzdřeny do jednotlivých objektů)

- výhoda strukturovaných metodik
  - mohou být jednodušší, vytvářené modely mohou být srozumitelné zákazníkovi => zákazník se snáze účastní strukturované analýzy (??)
  - návrh systému může být rychlejší (nevytváříme přídavnou strukturu tříd)

- nevýhody strukturovaných metodik
  - jsou považovány za nemoderní
  - výsledný systém se většinou hůře udržuje
  - nižší stupeň abstrakce -> složitější analýza, návrh,… velkých systémů

- tvorba modelu = abstrakce klíčových vlastností studovaného systému (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 kódujeme

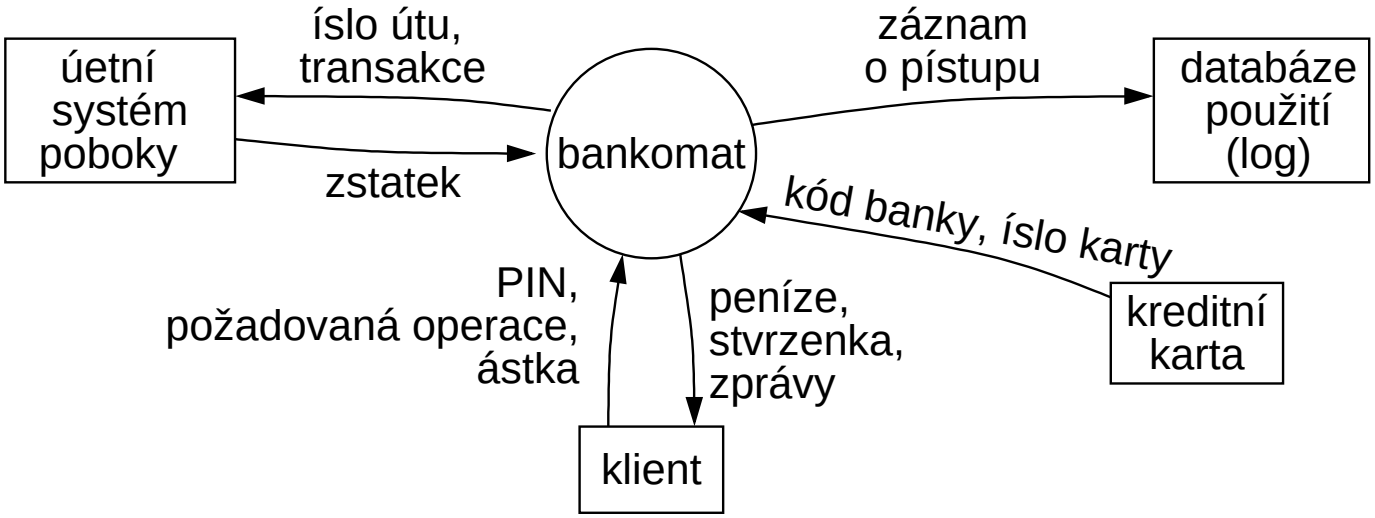
- vytváření modelu - podpora CASE nástrojů (editor modelů, částečná kontrola modelu, automatická tvorba dokumentace), srovnání některých CASE nástrojů podporujících strukturované metodiky v článku V. Řepy: Programování ve velkém, Softwarové noviny, 5/2003.

Model kontextu systému

- hranice systému, tj. co bude tvořit systém a co bude okolí systému
- v některých případech nemusí být úplně zřejmé, 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 - kontext diagram)
- hranice vytvářeného systému - kolečko s vepsaným názvem systému
- lidé, organizace nebo jiné systémy, se kterými náš systém komunikuje (terminátory) - pojmenovaným obdélník
- vstupující a vystupující data - šipky mezi systémem a terminátorem

Př. model kontextu systému pro bankomat:



Diagramy datových toků (data-flow diagram, DFD)

- také data-flow graph, work flow diagram, function model, bubble chart, process model...
- součást mnoha strukturovaných metod cca 1955-1990 (ale i některých OO metodik, např. OMT), různé notace
- jednoduché a intuitivní (je možné je vysvětlit zákazníkovi)
- použitelné pro modelování toku dat SW systémem, dokonce i modelování toku dat a fyzických předmětů (materiálu) v organizaci

- data jsou zpracovávána posloupností kroků (kroky provádějí lidé, funkce programu)

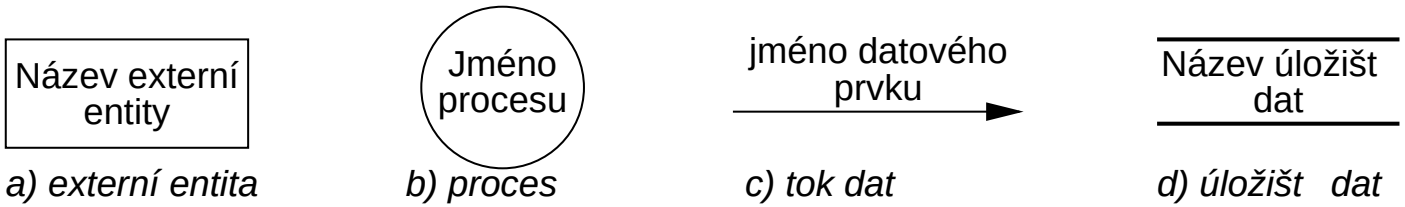
Základní notace pro DFD

- 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. (tj. odpovídá pojmu "aktér" z případu užití)
  - znázornění obdélníkem s názvem uvnitř

- proces
  - provádí transformaci dat
  - znázornění kolečkem (bublinou)
    - je vhodné bubliny číslovat
    - a je nutné bubliny 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

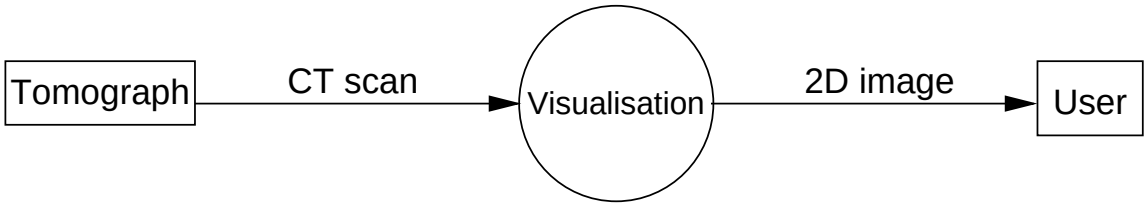


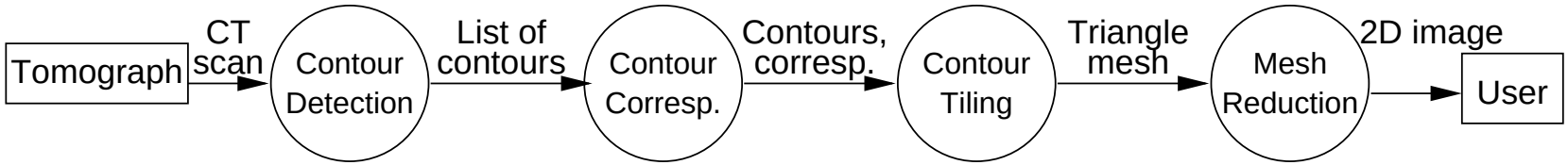
Používá se pro reprezentaci systému libovolné úrovně abstrakce

- DFD úrovně 0 = fundamentální model systému, model kontextu systému
  - celý SW systém je zakreslen jako jedna bublina, má jeden nebo více vstupů a výstupů

- DFD dalších úrovní
  - systém rozdělíme do menších částí a znázorníme na větší úrovni podrobnosti (model je postupně zjemňován)
  - vytvořené DFD by neměly být příliš velké - měly by se bez problémů vejít na papír velikosti A4

Př. Vizualizace snímků z tomografu





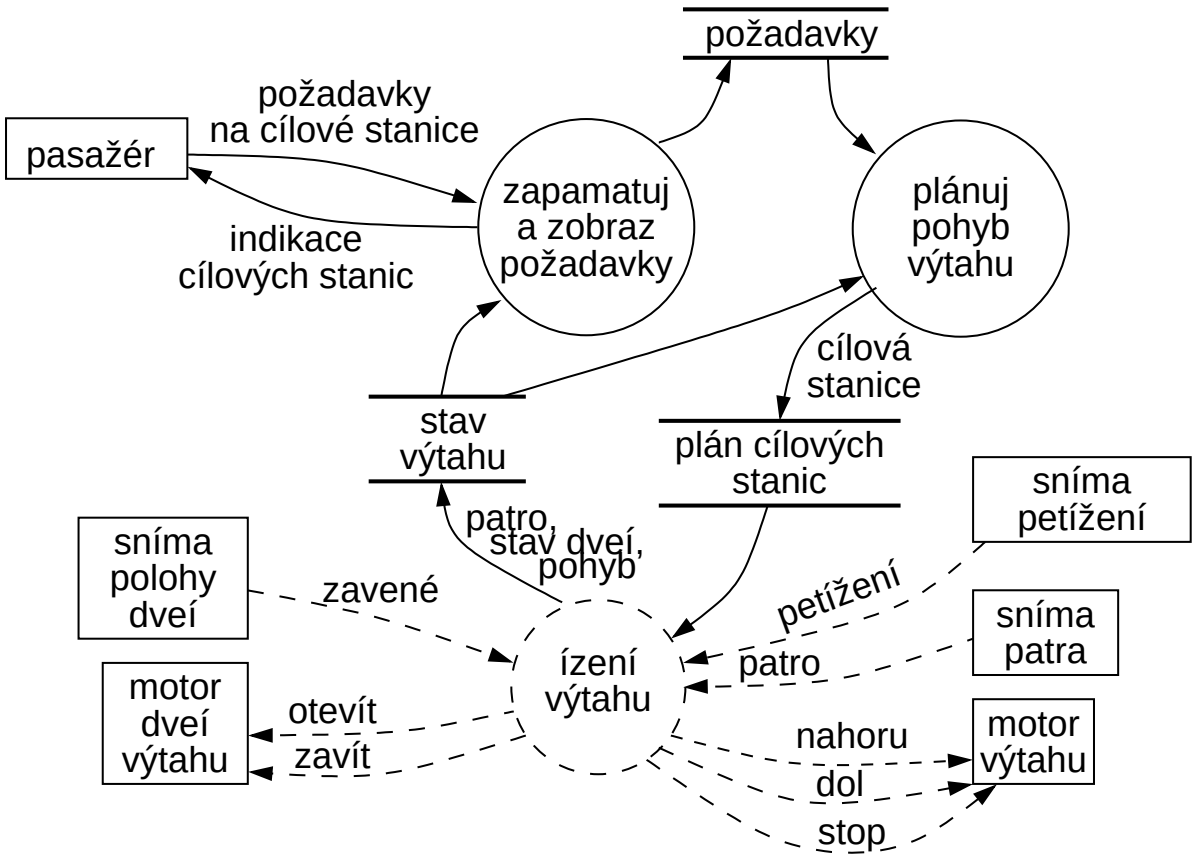
Co by se v DFD nemělo vyskytovat:

- černé díry - bubliny, které mají vstup, ale nemají výstup
- spontánní generátory - mají výstup, ale žádný vstup; jediný rozumný případ je generátor náhodných čísel
- neoznačené procesy a datové toky – vyskytují se zejména proto, že analytik není schopen najít pro ně rozumný název, např. sdružují nesouvisející procesy/toky dat (proces nebo datový tok je nutno rozdělit)

Rozšíření DFD

- výše uvedená základní varianta DFD je vhodná pro modelování systémů řízených datovými vstupy prakticky bez žádné vnější události (typicky obchodní aplikace).
- Pro RT systémy celá řada rozšíření základního DFD (tok řízení přerušovaná šipka a řídící proces bublina zakreslená přerušovanou čarou

Př. Řízení výtahu



Dále potřebujeme

- vytvořit definici dat
- specifikovat logiku procesů (na konečné úrovni zjemnění doplňujeme popis specifikací procesu buď v přirozeném jazyce, nebo v pseudokódu)

Datový slovník

- metoda reprezentace obsahu dat v datových tocích a pamětech
- seznam datových prvků s definicemi

Yourdonův "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.

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 s dodatky typu "Karel IV."?

-> formálnější definice v datovém slovníku, data

- jednoduchá - známé typy dat, je třeba zadat obor hodnot, příp. použité jednotky, příp. přesnost
- složená - popíšeme odkazem na jednoduché položky

Pak např. Jméno osoby je 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. | Ph.D.]
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 | ®]
malé_písmeno = *malá písmena české abecedy*
[a | á | b | c | č | ... | z | ž]

- 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 datového slovníku 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"

- dvě (stejné) definice, je nebezpečí, že při změně jednu z nich zapomeneme opravit

ERA diagramy

- popisují strukturu uchovávaných dat na vysoké úrovni abstrakce
- základy (Chen 1976), později vývoj různými směry - existuje mnoho notací,
- slouží jako vstup pro návrh databáze, např. "pamětí" v DFD

- podrobně viz Databázové systémy

nebo např.

Jaroslav Pokorný: Konstrukce databázových systémů.
Vydavatelství ČVUT, 1999.

Specifikace činnosti procesů

- zjemňováním DFD - rozklad problému na elementární procesy komunikující pomocí datových toků
- nutné specifikovat, co se děje v elementárním procesu

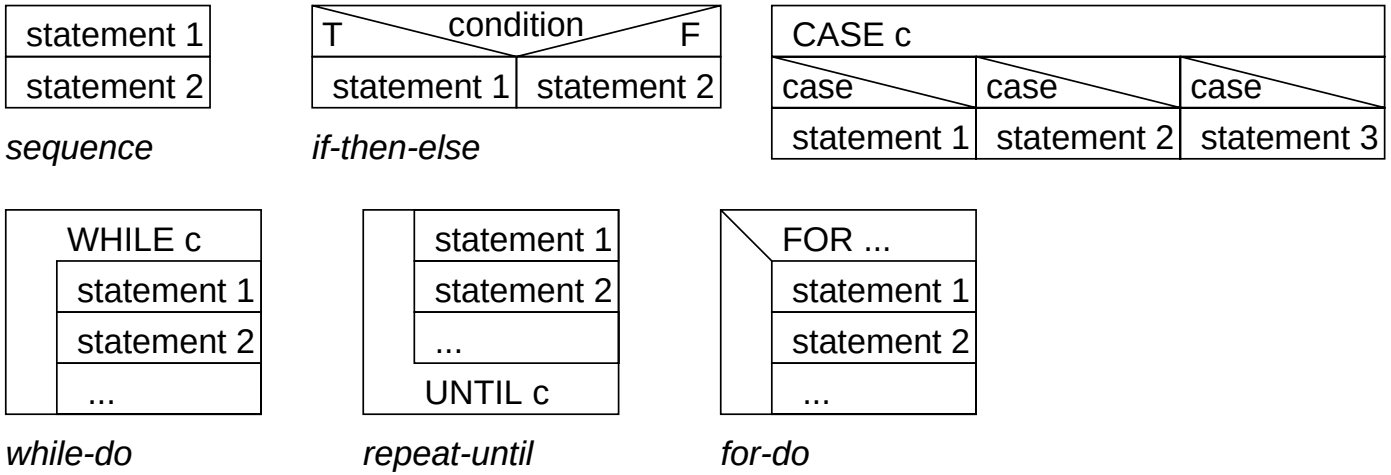
- použití nástrojů:
    - pseudokódy nebo jejich grafický ekvivalent
    - rozhodovací tabulky
    - rozhodovací stromy
    - konečný automat (konvence stavového diagramu z UML)

Pseudokódy

- základní myšlenka - omezený slovník běžného jazyka, přidáme strukturu programovacího jazyka
- slovník se skládá z:
    - příkazů
    - rozhodovacích konstrukcí
    - opakovacích konstrukcí
- 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, proces pravděpodobně není elementární

Nassi-Shneidermanovy diagramy

- někdy se používají místo pseudokódu
- po vysvětlení jsou pro zákazníky často srozumitelnější než pseudokód (podle publikovaných výzkumů pro 75-80% lidí)
- vyžadují nemalé množství grafiky - vyplatí se pouze tehdy, máme-li SW podporu pro jejich vytváření
- neměli bychom překročit 3 úrovně vnoření
-



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

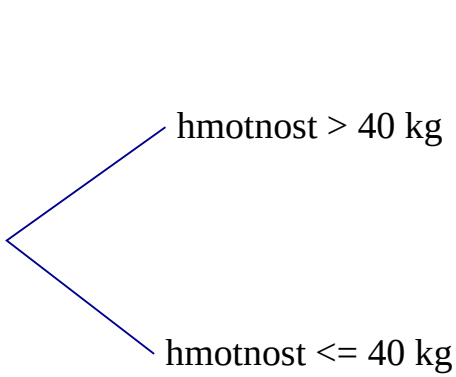
- používají se, pokud by byl pseudokód vyjadřující podmínku složitý
- najdeme všechny relevantní vstupy, určíme počet možných kombinací
- vytvoříme tabulku
    - sloupce = pravidla: počet pravidel je dán počtem kombinací vstupů
    - řádky tabulky: nejprve všechny vstupy, pak všechny výstupy

- horní polovina tabulky definuje podmínky
- ve spodní polovině je pro každý sloupec (tj. podmínku) uveden výstup

- někdy bývá srozumitelnější rozhodovací strom:

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

Př. Rozhodovací strom



	Lék1	Lék2
v k > 18	A	
vk <= 18	A	
vk > 18		A
vk <= 18		A
vk > 18		A
vk <= 18	A	
vk > 18	A	A
vk <= 18	A	

Vyvažování modelů

strukturovaná analýza používá viz výše:

- 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 - modelují funkce systému
    - datový slovník, ERA diagramy - modely pro 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:
  - 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.

Strukturovaná analýza

- využití nástrojů uvedených výše

dvě známé metodiky:

- klasická DeMarcova metodologie
- Yourdonova "moderní strukturovaná analýza"

Klasická DeMarcova metodologie

- vstupem uživatelské požadavky, výstupem 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

Co má analytik vyrobit – model původního systému nebo nového?

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

- vytváření 4 modelů systému:

1. fyzický model stávajícího systému (jaký systém používá zákazník?)
   - analytik zmapuje stávající systém, jeho funkční strukturu a data
   - model může obsahovat zpracování 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, tj. modelujeme 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 funkce
   - 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 (což je koneckonců pravda)
- 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 "oddechu" prodlužuje, což snižuje ochotu spolupracovat s analytikem (tj. 4 modely je prostě moc)
- zmíněné problémy řeší "moderní strukturovaná analýza" (Yourdon 1989)

Moderní strukturovaná analýza

- místo čtyř modelů systému se zaměřuje přímo na nalezení esenciálního modelu systému, tj. logického modelu nového systému

- kroky:
  - vytvoříme model prostředí (definuje hranici mezi systémem a zbytkem světa)
    - definujeme účel systému (ne delší než odstavec)
    - vytvoříme model kontextu systému
    - vytvoříme počáteční datový slovník definující data putující mezi systémem a terminátory
  - vytvoříme seznam událostí
  - na základě událostí vytvoříme předběžný model chování systému
  - model chování systému přestrukturujeme do konečného modelu chování (= esenciální model systému)

- vytvoříme uživatelský implementační model (doplňuje esenciální model o informace nutné pro implementaci modelu)
- konec analýzy, následuje návrh architektury, podrobný návrh a kódování

Vytvoření seznamu událostí

- události se klasifikují na:
  - o událost datového toku (F) – událost, která se projeví příchodem dat; např. událost "přišla objednávka"
  - o časová událost (T) - např. zpracování transakcí mezi bankovními účty nastane ve 3:00
  - o řídící událost (C) - asynchronní událost např. v RT systémech

- identifikace událostí: procházíme každý terminátor, ptáme se jaké akce může provádět nad systémem (obvykle probíráme společně s uživateli, kteří hrají role terminátorů - analogicky jako při identifikaci případů užití v OO analýze)

- pro každého kandidáta na událost se ptáme, zda je skutečně událostí, tj. zda v jejím důsledku systém vyprodukuje výstup nebo změní svůj stav

- pro kandidáta na událost se ptáme, zda se všechny instance události týkají stejných dat (pokud ne, budou to nejspíš dvě různé události; cílem je rozlišit mezi různými událostmi, které se náhodou dějí společně nebo vypadají podobně)

- pro každou událost ptáme: "musí systém nějak reagovat, pokud se událost neudá podle očekávání?" (tj. modelujeme odpovědi na chyby/poruchy terminátorů; např. zboží nepřijde v očekávané době, co musí systém udělat?)

Vytvoření předběžného modelu chování

- ne postup shora-dolů, ale identifikace odpovědí na události:
  - pro každou událost ze seznamu nakreslíme bublinu, očíslujeme jí podle čísla události
  - bublinu pojmenujeme podle odpovědi na událost (např. odpověď na "zákazník zaplatil fakturu" je "vlož platbu mezi příjmy")
  - pokud je na událost více odpovědí, přidáme pro další nezávislou odpověď další bublinu (nezávislé odpovědi = mezi bublinami není komunikace)
  - k bublině nakreslíme vstupy, výstupy a potřebné paměti
  - identické bubliny sdružíme do jedné (identické bubliny mají stejný vstup, výstup a proces; např. různé typy objednávek mohou mít stejnou odpověď)
  - výsledný DFD zkontrolujeme oproti kontextovému diagramu a seznamu událostí

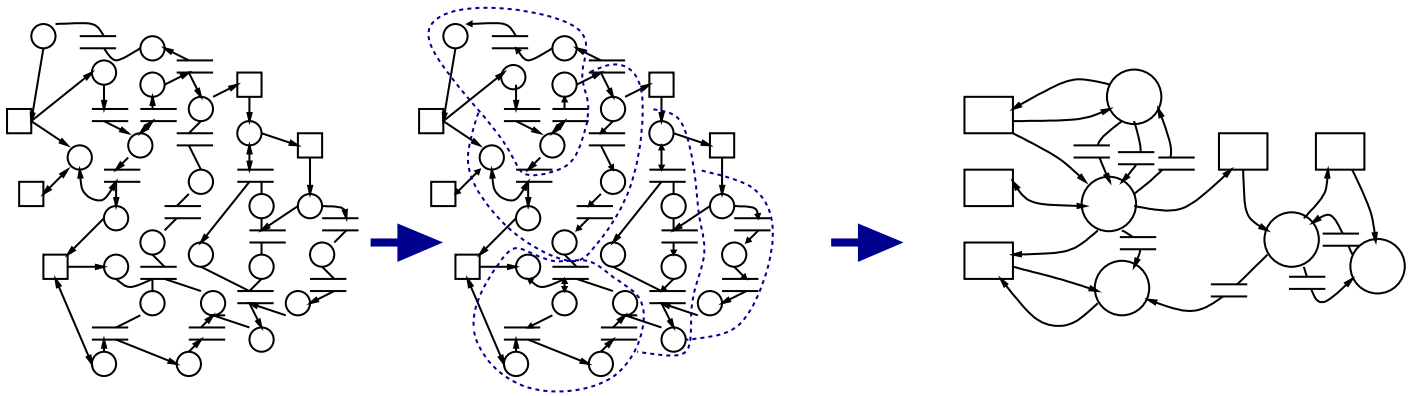
- výsledný model by měl:
    - popisovat pouze logické procesy a podstatu transformace dat
    - zcela vynechat implementaci (nerozlišuje ani, zda funkci provádí člověk nebo počítačový systém) - proto vynecháme např.:
        - procesy, jejichž účelem je přenos dat z jedné části systému do jiné
        - procesy pro verifikaci dat vzniklých uvnitř systému
        - procesy pro fyzický vstup a fyzický výstup ("vytiskni fakturu")

- je třeba rozlišovat mezi zdrojem informace ("obchodní zástupce") a mechanismem pro vstup/výstup ("systém pro zadávání objednávek"); mechanismy vynecháme

Dokončení modelu chování

- předběžný DFD chování systému bude zbytečně komplikovaný, zjednodušíme ho
- dokončíme specifikaci procesů
- dokončíme datový slovník

- zjednodušení DFD chování systému strukturováním:
  - agregace = spřízněné procesy sdružíme, budou tvořit bubliny v DFD vyšší úrovně
  - každý agregát se týká úzce příbuzných odpovědí na události, tj. většinou obsahuje procesy zpracovávajících příbuzná data
  - sdružovat bychom měli skupiny po cca 7+-2 procesech+pamětech, při tom máme příležitost "schovat" paměti, které jiné procesy nepotřebují



- někdy procesy naopak nejsou primitivní a vyžadují další rozdělení (většinou jsme bublinu nedokázali dobře pojmenovat, proto jí rozdělíme na primitivní procesy)

- vytvoříme specifikaci procesů
- dokončíme datový slovník, ERA model

- výše uvedené informace tvoří tzv. esenciální model systému

Vytvoření uživatelského implementačního modelu

- alokace esenciálního modelu na lidi a stroje (které bubliny a paměti budou realizovány manuálně) - rozhoduje uživatel
- rozhraní aplikace s uživatelem (volba vstupních a výstupních zařízení, formáty obrazovek, formáty výstupů)
- omezení, např. objemy dat, časy odpovědí na různé události atd., tj. mimofunkční požadavky na aplikaci

- oficiálně poslední krok moderní strukturované analýzy požadavků.
- následuje návrh architektury, podrobný návrh, implementace, testování

- při analýze nemusejí být použity všechny nástroje (modely) - používají se jen ty, které mají v daném kontextu smysl

- další metodiky pro strukturovanou analýzu:

  - např. SADT, SREM/RDD, SA/SD.
  - v ČR asi nejpoužívanější SSADM (Structured Systems Analysis and Design Method); podobný záběr a nástroje jako DeMarcova a Yourdonova strukturovaná analýza (výstupy - DFD, model entit, ...), je to standard - velmi podrobně definované kroky včetně výstupů a předepsaných kontrol před přechodem k dalšímu kroku.
  - použití SSADM v některých zemích podmínkou pro získání státních zakázek.