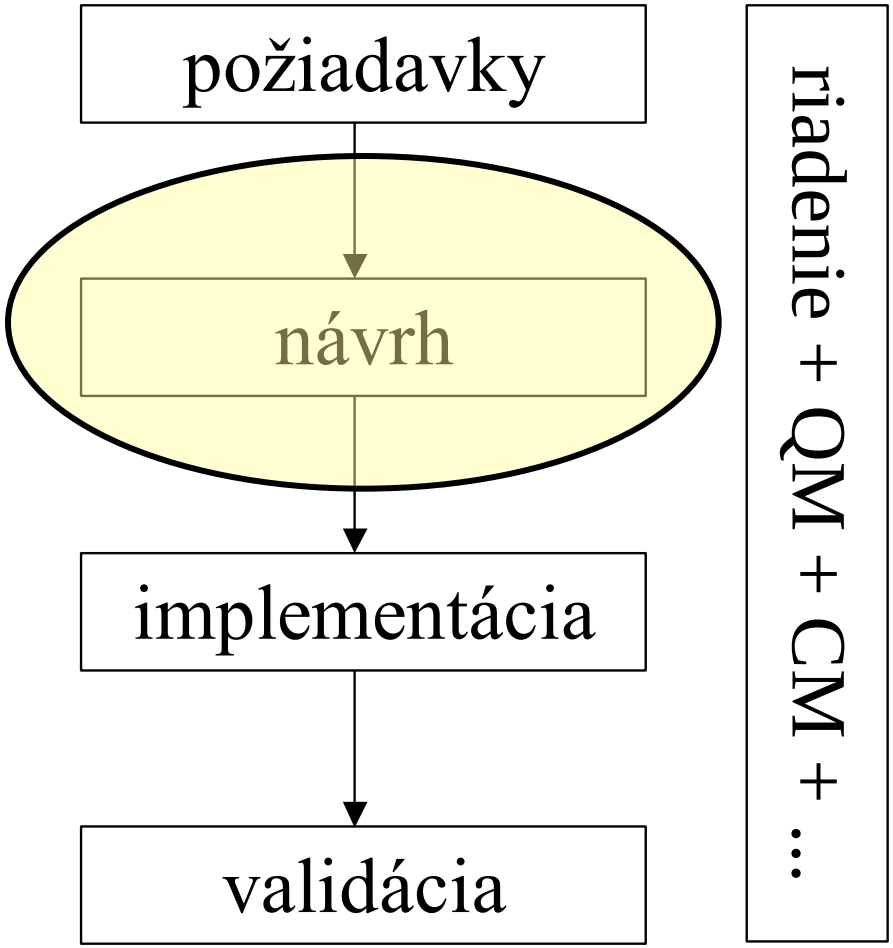
Cvičenie z PTS

23.3.2010

Návrh systému



Návrh

- hlavným cieľom je určiť, **ako** bude daný SW produkt realizovaný
- hlavný vstup: **špecifikácia požiadaviek**
- hlavný výstup: **návrh (ako dokument)** – slúži najmä ako zadanie pre programátorov

Čo obsahuje návrh (dokument)

- štruktúru systému na hrubej i jemnej úrovni
  - komponenty a vzťahy, resp. rozhrania medzi nimi
- určenie HW/SW platformy
- popis používateľského rozhrania
- popis použitých dátových štruktúr
- popis algoritmov
- ...

Podrobnosť návrhu závisí od konkrétnej situácie (veľkosť a zložitosť projektu, požadované atribúty kvality, použitý proces vývoja, ...)

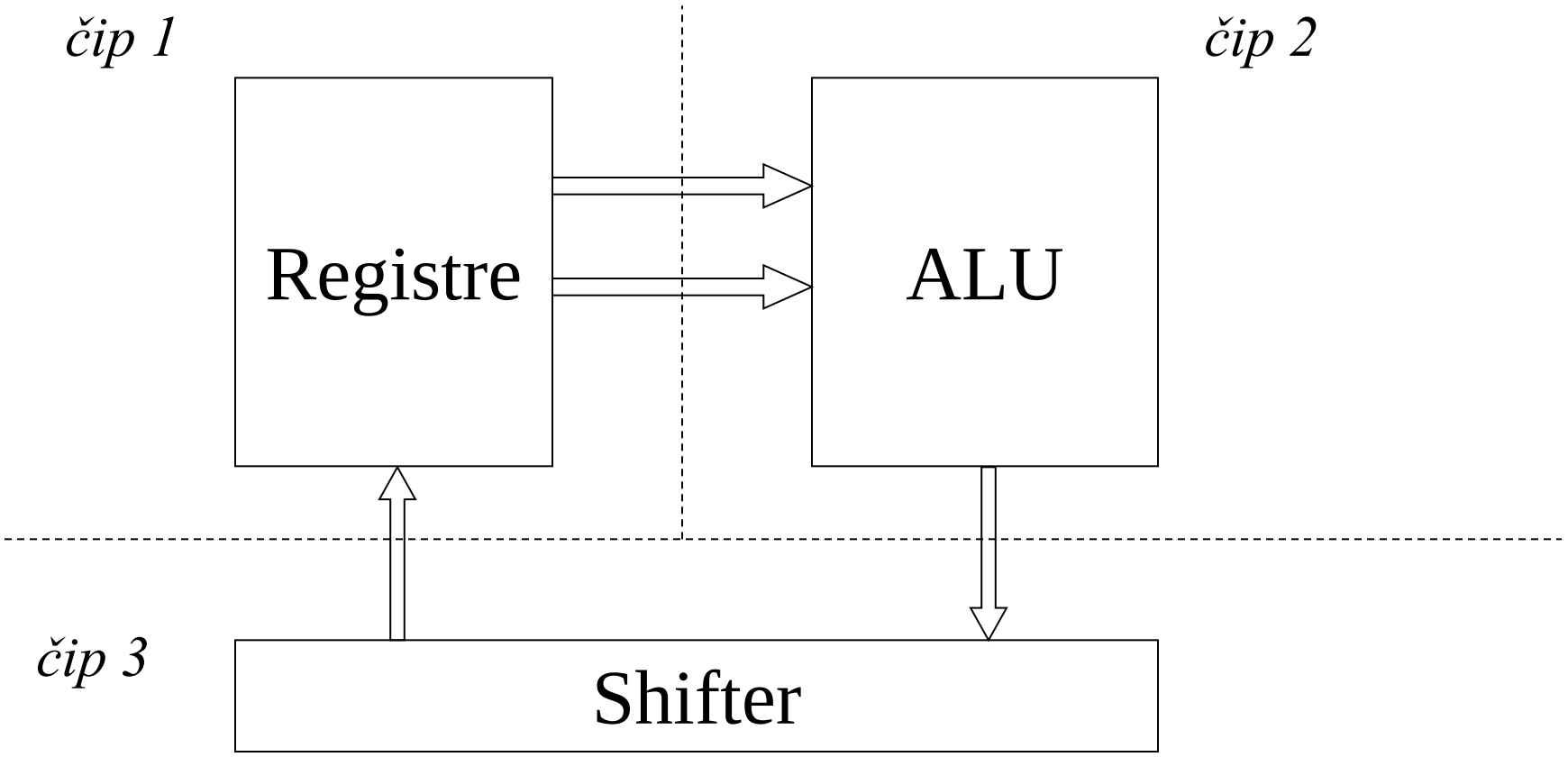
Architektúra a návrh

Návrh obsahuje aj iné než architektonické rozhodnutia,
napríklad:

- aké algoritmy a dátové štruktúry budú použité vnútri modulov
- vzhľad používateľského rozhrania

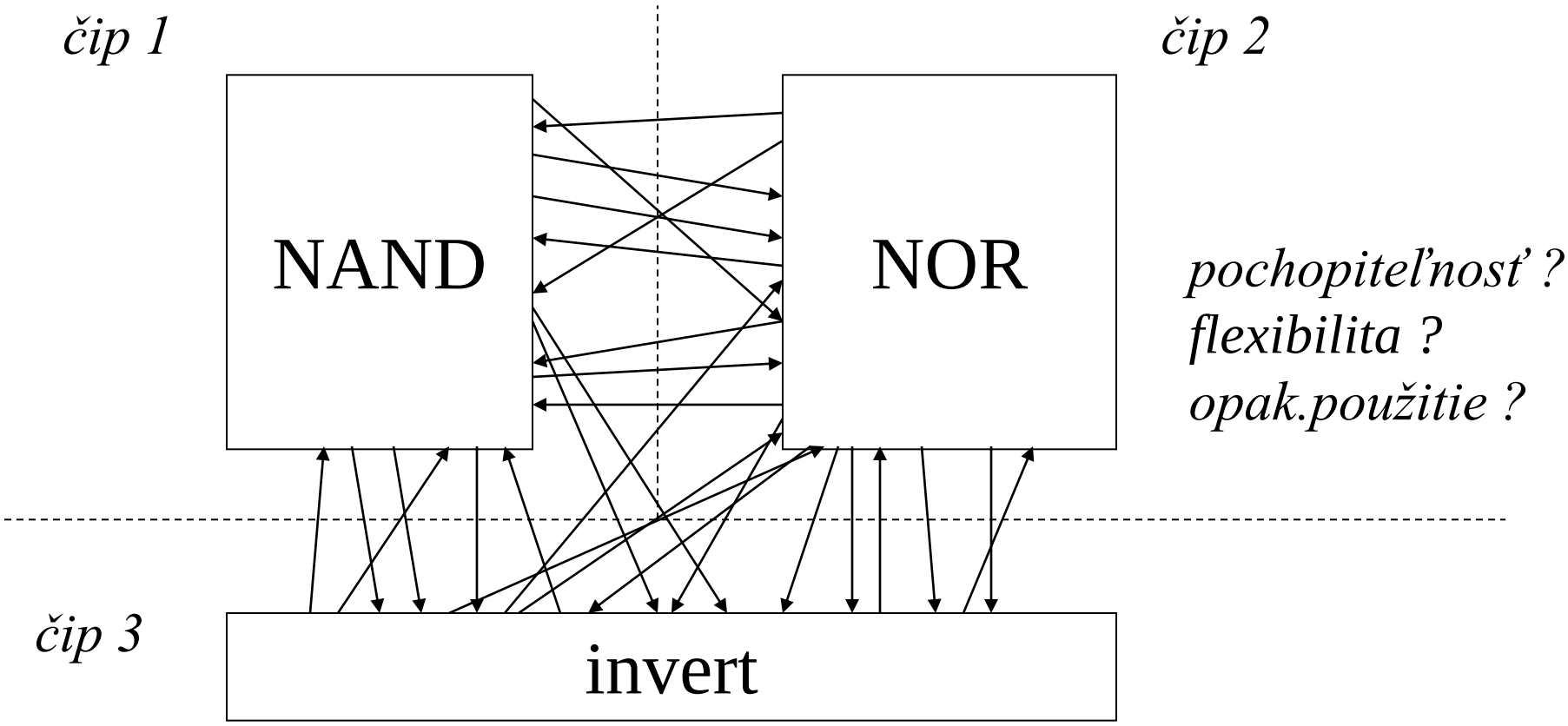
Motivačný príklad – CPU

architektúra č.1



Motivačný príklad – CPU

architektúra č.2



Dobrá architektúra

Definícia: **Stakeholder** (účastník procesu vývoja softvéru) je človek alebo organizácia, ktorý, resp. ktorá má priamy alebo nepriamy vplyv na požiadavky kladené na systém.

Napríklad: koncový používateľ, zákazník, vývojár, vývojová organizácia, oddelenie zodpovedné za prevádzku, údržbu, ...

Dobrá architektúra 2

Stakeholderi majú rôzne požiadavky, napríklad:

- **koncový používateľ**: funkčnosť, jednoduchosť použitia, výkon, bezpečnosť, spoľahlivosť !
- **zákazník:** cena, termín !
- **vývojová organizácia – vedenie:** cena, termín, primerané požiadavky na ľudské zdroje, zohľadnenie krátkodobých aj dlhodobých zámerov firmy !
- **vývojová organizácia – časť zodpovedná za údržbu:** flexibilnosť, udržiavateľnosť !
- ...

Dobrá architektúra umožní systému tieto požiadavky realizovať.

(sú principiálne protichodné a často nie sú explicitne sformulované)

Vytvorenie dobrej architektúry

- všetky požadované vlastnosti sa naraz optimalizovať **nedajú**
- nie všetky požiadavky sa odrazia na architektúre
- nasledujú jednotlivé požiadavky, v poradí
  - vlastnosti systému viditeľné počas behu
  - vlastnosti systému neviditeľné počas behu
  - ostatné požiadavky

Výkon (performance)

- výkon =
  - doba odozvy
  - počet operácií za jednotku času (priepustnosť)
  - škálovateľnosť

- **riešenie na architektonickej úrovni:**
  - minimalizovať komunikáciu medzi komponentmi (najmä ak sú na rôznych počítačoch, v rôznych procesoch a pod.)
  - rozloženie záťaže (horizontálne, vertikálne)

Bezpečnosť (security)

- bezpečnosť = schopnosť systému odolávať pokusom o neoprávneným prístup, resp. narušenie činnosti

- **riešenia na architektonickej úrovni:**
  - oddelenie funkcií kritických z pohľadu bezpečnosti (autentifikácia, autorizácia, ...) do samostatných komponentov, ktoré sú dobre oddelené a chránené a ktoré môžu byť overiteľne korektne implementované
  - obmedzenie komunikácie medzi systémom a jeho okolím
  - ...

Spoľahlivosť (availability, reliability)

- availability = pravdepodobnosť, že systém je v danom okamihu dostupný
- reliability = pravdepodobnosť, že systém dá správny výsledok (resp. že nezlyhá)

- **riešenia na architektonickej úrovni:**
  - redundantné komponenty
  - dobre riešená komunikácia o chybách a ich spracovanie
  - špeciálne komponenty na správu a monitorovanie (napr. watchdog timers)
  - ...

Funkčnosť (functionality)

- funkčnosť = systém poskytuje požadované funkcie (potrebné na plnenie úloh, pre riešenie ktorých bol vytvorený)

- táto vlastnosť v prevažnej miere nesúvisí s architektúrou

Jednoduchosť použitia (usability)

= ako rýchlo a ľahko sa používatelia naučia pracovať so systémom, do akej miery systém pomáha vyhnúť sa bežným používateľským chybám, či sa dá so systémom pracovať efektívne ...

táto vlastnosť v prevažnej miere nesúvisí s architektúrou

- vyplýva najmä z kvality používateľského rozhrania

Modifikovateľnosť (modifiability)

- modifikovateľnosť = ako ľahko sa v systéme robia zmeny (pridanie, odobranie, resp. zmena funkčnosti, prispôsobenie sa zmenám prostredia, reštrukturalizácia, oprava chýb)

- **podstatne závisí od architektúry**
  - námaha potrebná na vykonanie zmeny závisí najmä od toho, akú veľkú časť systému je potrebné upraviť
    - jeden modul
    - viac modulov
    - potrebný je zásah do architektúry

Modifikovateľnosť 2

Riešenie na architektonickej úrovni:

- izolovať miesta možných zmien do samostatných komponentov (a takto ich oddeliť od zvyšku systému príslušným rozhraním)

Modifikovateľnosť 3

Ako predpovedať možné zmeny ?

- úvahou
- skúsenosťami s podobnými (už existujúcimi) systémami
- podľa neistých, resp. nejednoznačných miest v špecifikácii
- predpokladať, že celá funkčnosť nebude realizovaná

Portabilita

- portabilita = ako ľahko je systém upraviť tak, aby pracoval v inom (hardvérovo-softvérovom) prostredí
- špeciálny prípad modifikovateľnosti

- **riešenie na architektonickej úrovni:** uzavrieť kód špecifický pre platformu do jedného (resp. niekoľko málo) modulov – „portability layer“

Opakovaná použiteľnosť (reusability)

- reusability = časti systému resp. jeho architektúra môžu byť opakovane použité v budúcich systémoch

- **riešenie na architektonickej úrovni:** voľné väzby medzi komponentmi

Integrovateľnosť (integrability)

- integrovateľnosť = schopnosť poskladať samostatne vyvinuté komponenty do fungujúceho celku (resp. možnosť použiť pri vytváraní systému už existujúce komponenty)

- riešenie na architektonickej úrovni:
  - zníženie externej zložitosti komponentov a interakčných mechanizmov
  - jasné rozdelenie zodpovednosti medzi komponenty
  - (pri existujúcich komponentoch samozrejme prispôsobenie sa ich rozhraniam)

súvisiaca vlastnosť: interoperabilita

Ďalšie faktory

- rýchlosť vytvorenia (time to market)
  - napr. výraznejším použitím existujúcich komponentov
- cena
  - závisí od architektúry napr. tým, aké technológie sa použijú
- očakávaná životnosť systému
  - ak dlhá – podstatná je modifikovateľnosť a portabilita (ovplyvňuje cenu a rýchlosť vytvorenia)

Zhrnutie / ostatné aspekty

- dobre definované komponenty
- skrývanie informácií, rozdelenie záujmov
- komponenty podľa programátorských tímov / špecializácií
- oddelenie aplikačnej logiky od infraštruktúry
- konzistentnosť komunikácie medzi komponentmi

- **JEDNODUCHOSŤ (KISS)**
- správne stanovené požiadavky
- overené riešenia

Vzory

Definícia: Vzor (pattern) je vopred pripravená množina návrhových (resp. analytických alebo implementačných) rozhodnutí vhodných pre istú situáciu, o ktorej sú známe určité vlastnosti. Túto množinu rozhodnutí je možné ako celok prevziať a po príslušnej úprave použiť.

Vzory existujú na rôznych úrovniach:

- architektonické vzory (štýly)
- návrhové vzory
- implementačné vzory (idiómy)
- analytické vzory

Súčasťou vzoru je obyčajne:

- názov
- popis problému
- popis riešenia
- dôsledky aplikovania riešenia (vlastnosti riešenia)

Architektonický štýl

Definícia: Architektonický štýl je vzor zaoberajúci sa popisom typov komponentov a spôsobu odovzdávania riadenia a údajov medzi nimi.

-- Bass, Clements, Kazman, 1998

- štýl určuje množinu architektúr, ktoré mu zodpovedajú
- štýl je určený
  - typmi a vlastnosťami komponentov (proces, procedúra, ...)
  - typmi a vlastnosťami konektorov, ktoré odovzdávajú riadenie a údaje ([vzdialené] volanie procedúry, prúd údajov, ...)
  - syntaktickými a sémantickými obmedzeniami týkajúcimi sa spájania komponentov konektormi

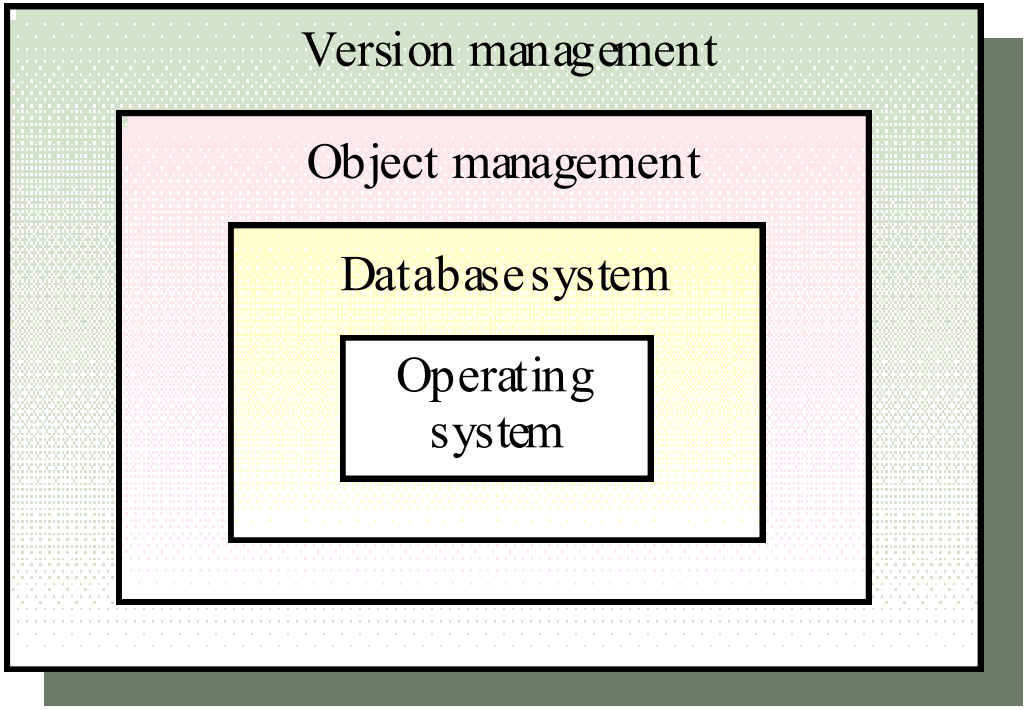
Príklady arch. štýlov

- Data-Centered
  - Repository
  - Blackboard
- Data Flow
  - Batch Sequential
  - Pipes and Filters
- Call and Return
  - Main Program and Subroutine
  - Object Oriented
  - Layered
- Virtual Machine
  - Interpreter
  - Rule-Based System
- Independent Components
  - Communicating Processes
  - Event Systems
    - Implicit Invocation
    - Explicit Invocation

tieto štýly **nie sú** disjunktné !

Príklad č. 1: štýl Layered

- **organizuje systém do postupnosti vrstiev, z ktorých každá poskytuje definovanú množinu služieb**
- vrstva N je implementovaná pomocou služieb vrstvy N-1 (alternatívne: pomocou služieb vrstiev 1 až N-1)
- dá sa použiť s rôznymi typmi komponentov a konektorov

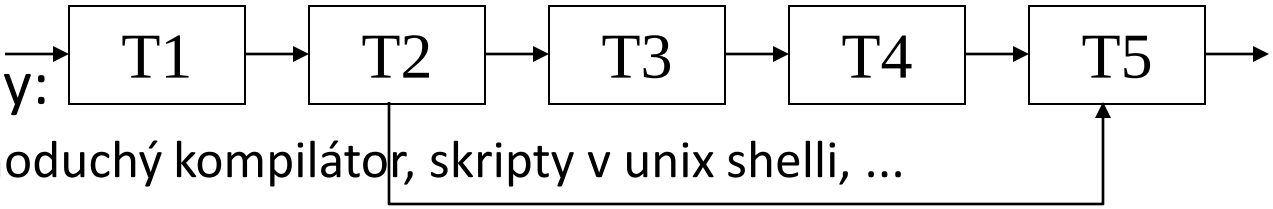


Iné príklady:

niektoré operačné systémy, OSI Reference Model, ...

Príklad č. 2: Pipes and Filters

- **popisuje systém ako sériu postupných transformácií údajov**
- komponenty:
  - filtre – transformujú vstupné údaje na výstupné, nemajú trvalý vnútorný stav, majú len minimálne väzby na okolie, bežia (navzájom) asynchrónne
- konektory:
  - pipes – bezstavové „rúry“, ktorými prechádzajú údaje
- príklady:
  - jednoduchý kompilátor, skripty v unix shelli, ...



Príklad č. 3: Repository

- **centrálnym prvkom je štruktúrovaný sklad údajov, s ktorým pracujú jednotliví (nezávislí) klienti**
- komponenty:
  - sklad údajov – je pasívny
  - klienti – majú navzájom nezávislé toky riadenia
- konektory:
  - dotazy a aktualizácie – ak sa odovzdáva riadenie skladu údajov, tak len za účelom vykonania príslušnej operácie (čítanie/zápis)
- príklady:
  - „klasický“ informačný systém
  - nástroje CASE založené na spoločnej databáze

Príklad použitia štýlu Repository:

CASE Toolset