1. Informační systém, jejich základní vlastnosti a typy

Informační systém (IS) je systém pro sběr, udržování, zpracování a poskytování informací.

Funkce informačního systému

Konkrétní procesy (činnosti) podporující základní cíle informačního systému:

- získávání informací
- zpracování informací (evidence, organizace – pořádání, kategorizace, konverze –změna média, třídění, vyhledávání, agregace, odvozování nových informací, dolování znalostí)
- uložení informací (zaznamenávání a archivace dat, datová úložiště a datové sklady)
- přenos informací (v rámci počítačových sítí)
- zpřístupnění informací (tisk, zobrazení, vizualizace, šíření…)

Životní cyklus IS

Posloupnost etap (činností) provázející IS od prvotní představy po vyřazení

Fáze životního cyklu IS: specifikace požadavků – analýza systému (tvorba modelů) – návrh (technické řešení) – Implementace (z návrhu udělat funkční aplikaci) – Testovní – nasazení a podpora

Následující body vystihují vlastnosti, které by kvalitní IS s maximální výkonností měl splňovat.

- Musí obsahovat **nutné informace**, které **uchovává, analyzuje** a s potřebnou rychlostí **předává procesům**. Dané informace se týkají zejména **vlastní činnosti firmy jako je výroba**, **evidence zákazníků**, zásob, zaměstnanců, finance, stav a vývoj vlastních výrobků.
- Musí obsahovat informace o konkurenci, světovém trhu, trendech výroby, optimalizaci výrobních procesů, o místech působnosti firmy, o strategických cílech a podobně.
- Musí obsahovat moduly pro zjednodušení a urychlení výroby, čímž je míněno hlavně urychlení a zefektivnění návrhu výrobků, technologická příprava výroby a její řízení.
- Musí umožňovat rychlou komunikaci pracovníků firmy, jednotlivých pracovních úseků, ale musí také zahrnovat komunikaci se světem.
- **Musí umožňovat z dostupných informací zpracovávat cíle a strategie firmy**, koordinovat činnost různých procesů a tím přispívat k zefektivnění činnosti firmy.
- Musí nabízet **rychlou komunikaci se zákazníkem** přes počítačovou síť.
- Musí obsahovat další nutné moduly k vedení **firmy jako jsou statistiky, mzdy, účetnictví**, kompletní **personalistika**, sklad, oblast manažer – marketing, výroba a další.

Typy informačních systémů

Podnikové informační systémy (BIS – business information system)

Systémy, provozované v kontextu **konkrétní organizace**, jejichž **účelem je správa informací a znalostí a jejich integrace do podnikových procesů.**

Obsažené informace jsou chápány jako jeden z ekonomických zdrojů (aktiv) organizace.

Rozlišují se systémy podporující:

- **vlastní činnosti a služby organizace** (automatizace podnikových procesů – např. CIM, workflow management, elektronický obchod, systémy pro tvorbu a správu dokumentů)
- **manažerské systémy, podporující řídící** a administrativní funkce. Jako softwarové vybavení se nabízejí zpravidla tzv. typová řešení pro konkrétní odvětví nebo obchodní model.

Provozní, transakční systémy

- **ERP –** enterprise resources planning: systémy na podporu provozu (chodu) firmy
  Technologický princip: aplikační software, OLTP (Online Transaction Processing), relační databáze.
  OLTP je technologie uložení dat v databázi, která umožňuje jejich co nejsnadnější a nejbezpečnější modifikaci ve více uživatelském prostředí. Jedná se o přístup používaný v současné době v převážné většině databázových aplikací.

Systémy na podporu plánování

- **SCM** – supply chain management: plánování dodavatelských logistických řetězců
- HR – human resources – řízení lidských zdrojů
- APS – advanced planning and scheduling: systémy na podporu vnitropodnikového (dílenského) plánování,

Systémy řízení vztahů se zákazníky

- **CRM** – customer relationship management: Shromažďování, zpracování a využití informací o zákaznících firmy za účelem poznat, pochopit a předvídat potřeby, přání a nákupní zvyklosti zákazníků. Podporuje komunikaci mezi firmou a jejími zákazníky.

Systémy na podporu rozhodování

- **BI** – business intelligence)
  Technologický princip: OLAP (Online Analytical Processing), datové sklady (data warehouse), dolování dat (data mining) – základem není realizace transakcí, ale prohledávání a analýza velkých objemů dat
- MIS – management information system,
- EIS – executive information system,
- DSS – decision support system,

Systémy pro tvorbu a správu dokumentů

- **DMS** – document management system
  Systémy umožňující efektivní práci s elektronickými dokumenty a jejich obsahem v průběhu celého jejich životního cyklu.
  Typickými procesy jsou tvorba, schvalování, evidence, digitalizace, prohlížení, editace, publikování, komunikace, sdílení, uložení, vyhledání, archivace, skartace apod.

  Obvykle je zahrnuta i skupinová spolupráce, workflow management a propojení dokumentů s informacemi v ostatních (např. provozních) informačních systémech.

  Technologický princip: aplikační software, obsahující nástroje pro tvorbu, publikování, fulltextové vyhledávání, řízení přístupu k elektronickým dokumentům, správu verzí, sledování historie použití a změn.

- DTP – desktop publishing

Nepodnikové systémy

Knihovní systémy

Systémy určené k automatizaci procesů realizovaných v knihovně. Obvykle mají modulární strukturu; typické moduly jsou akvizice, katalogizace, výpůjčky apod. Zpravidla obsahuje i nástroje pro zapojení do sítě knihoven a pro komunikaci s externími zdroji.
Technologický princip: aplikační software **provozního (transakčního) typu**,

Geografické informační systémy (GIS)

Prostorově orientované informační systémy, provozované za podpory informačních a komunikačních technologií. **Datovou základnu tvoří digitální geografické informace** ve formě záznamů nebo objektů (tzv. geoprvky), s nimiž specializovaný software umožňuje provádět manipulaci (zápis a editace údajů, uložení, vyhledávání, propojování, transformace a vizualizace**), lokalizaci (určení polohy), geografické analýzy a modelování (např. trojrozměrný model terénu).**

Expertní systémy

Počítačové aplikace nebo **systémy simulující poznávací a rozhodovací činnost experta při řešení složitých úloh s cílem dosáhnout ve zvolené problémové oblasti kvality rozhodování na úrovni experta.**
Technologický princip: základní součásti tvoří báze znalostí, báze dat (faktů) k řešeným případům a řídící mechanismus (inferenční neboli odvozovací stroj, rozhodovací jádro), tj. program pro práci s těmito bázemi využívající technik umělé inteligence. Tyto základní součásti obvykle doplňuje modul pro komunikaci s uživatelem

Systémy na podporu návrhu a projekční činnosti

Jde o velkou rychle se rozvíjející **oblast IT, která zastřešuje širokou škálu činnost navrhování a kreativní lidské činnosti s pomocí počítače.**

Podstata: počítačová podpora (automatizace) některých procesů (návrh, výroba ap.)

- **CAD** (computer-aided design): **počítačem podporované projektování**.
- CAM (computer-aided manufacturing): strojírenství
- AEC (Architecture-Engineering-Construction), BIM (Buliding Information Model), CAAD (Computer-aided architectural design): stavebnictví a architektura
- **CASE** (Computer-aided software engineering): **počítačem podporované procesy tvorby software.** Budeme se jim podrobněji věnovat

Veřejné Informační systémy

Informační systémy, které jsou dostupné široké veřejnosti a poskytují veřejné informační služby. V tomto smyslu se jedná o jakékoli informační systémy bez ohledu na jejich provozovatele, obsah, typ, formu a příp. cenu poskytovaných informací a služeb. Opakem jsou tzv. privátní, uzavřené, neveřejné informační systémy (např. podnikové informační systémy, systémy zajišťující obranu státu, osobní informační systémy ad.).

Státní informační systém, informační systém veřejné správy

Systém, jehož účelem je **podporovat činnosti provozované při výkonu veřejné správy,** tj. státní správy a samosprávy, a **poskytovat veřejné informační služby** včetně informací o subjektech veřejné správy. Představuje komplex navzájem propojených subsystémů, členěných z hlediska věcného, resortního a regionálního.
Nejdůležitější součást datové základny tvoří evidence (registry) základních skutečností nezbytných pro výkon veřejné správy: evidence obyvatel, evidence ekonomických subjektů, evidence území a územních jednotek.

eGovernment

Moderního, přátelského a efektivního úřadu

eHealth

Elektronické zdravotnictví (eHealth) je souhrnný název pro řadu nástrojů založených na informačních a komunikačních technologiích, které **podporují a zlepšují prevenci, diagnostiku, léčbu, sledování a řízení zdraví a životního stylu.**

eLibrary

Rozdělení dle technologie zpracování dat

Jedním z kriterií je rozdělení IS dle technologie zpracování dat na transakční (operativní, provozní) - OLTP (Online Transaction Processing) a analytické - OLAP (Online Analytical Processing).

- OLTP - umožňují skupině uživatelů vykonávat bezprostředně (online) velké množství transakcí
  - relační databáze
- OLAP - analýza velkého množství údajů, většinou jen pro čteni, nadstavba OLTP
  - např. BI

Cílem OLTP systémů je umožnit skupině uživatelů vykonávat bezprostředně (online) velké množství transakcí, tj. umožnit co nejsnadnější a nejbezpečnější modifikaci uložených dat ve více-uživatelském prostředí.
Jedná se např. o podnikové, bankovní, obchodní a výrobní systémy.
Cílem OLAP systémů je analýza velkého množství uložených údajů. Výsledky analýzy jsou souhrny a reporty, obecně informace získané z uložených dat, které slouží jako podklady pro rozhodování v oblasti řízení firem, či ekonomických a technologických procesů. Pro získání těchto informací je potřeba vykonat velké množství rozsáhlých výpočtů a agregací a to pružně a rychle (online).

Data uchovávaná v OLTP databázovém systému jsou periodicky zpracovávána (agregována) a poté ukládána do datového skladu, nad nímž se posléze provádí analýzy. Datový sklad je na rozdíl od OLTP databáze určen výhradně ke čtení dat pro potřeby nejrůznějších analýz.

Rozdělení IS dle uživatelů

- **Veřejné informační systémy** - Informační systémy, které jsou dostupné široké veřejnosti a poskytují veřejné informační služby. V tomto smyslu se jedná o jakékoli informační systémy bez ohledu na jejich provozovatele, obsah, typ, formu a příp. cenu poskytovaných informací a služeb.
- **Privátní, uzavřené,** neveřejné informační systémy (např. podnikové informační systémy, systémy zajišťující obranu státu, osobní informační systémy ad.).

2. Analýza informačních systémů (IS), role modelování a metodik při tvorbě IS

Informační systém

IS je systém pro sběr, udržování, zpracování a poskytování informací.

Informační systémy jsou založené na informačních a komunikačních technologiích. V poslední době se používá zkratka ICT (Information and Communication Technologies).

IS obsahuje data, znalosti a informace

- *Data* - jakékoli vyjádření (reprezentace) skutečnosti, schopné přenosu, uchování, interpretace či zpracování. Umožňují přenášet a zpracovávat odraz skutečnosti.
- *Znalosti* - výsledek poznávacího procesu, předpoklad uvědomělé činnosti. To, co jednotlivec ví po osvojení dat a po jejich začlenění do souvislostí. Účel znalostí - porozumět realitě.
- *Informace* - je definovaná pomocí dat a znalostí - data, která mají smysl (význam), sdělitelné (komunikovatelné) znalosti.

Znalost x Informace x Data

- **Data** = symboly, objektivní hodnoty, fakta vnímatelné smysly
- **Informace** = data interpretovaná s ohledem na význam a důležitost
- **Znalost** = interakce dat a informací se zkušeností, dovednostmi, hodnotami, myšlením člověka, ostatními fakty, informacemi a znalostmi
  – „informace + X“

Znalosti – data mining
Informace – OLAP/reporting
Data - OLPT

Pokud subjekt (příjemce) rozumí významu, který je ukrytý v datech a má o těchto datech jisté znalosti, znamenají pro něj tato data také nějakou informaci. **Znalosti jsou potřebné pro konverzi dat do informace.**

Informace představuje vypovídací schopnost dat, vzniká zpracováním dat a je cílem tohoto zpracování.

Role modelování a metodiky pří tvorbě IS

Složitost systému se promítá do složitosti jeho návrhu a realizace. (tedy i do modelování)

Fenomém složitosti

Roste **složitost používaných informačních systémů**. V důsledku rychlého vývoje technologií a stále rostoucí složitosti IS tradiční postupy návrhu selhávají. Složitost systému se promítá do složitosti jeho návrhu a realizace, ale i vlastního řízení projektů.

Složitost versus jednoduchost: pokud má být IS pro uživatele srozumitelný, jednoduše, intuitivně použitelný, je vnitřně složitý a naopak. (Připravit si stručnou přednášku je velmi pracné).

!Základní teze: rozsáhlé IS nelze začít rovnou „programovat“, lepit z malých, jednotlivých částí celek. Instinkty selhávají, optimismus je nezdravý.

Tvorba IS ad-hoc (lepením, chaoticky, bez plánu, bez analýzy) vede k tomu, že vytvořený systém:

- dělá něco jiného než by měl
- problémy se **řeší lokálně** - v závislosti na právě realizované části. Důsledkem je, že jedna a tatáž věc je řešena na různých místech **a po každé jinak.**
- opravy a změny jsou velmi obtížné a drahé. Jestliže opravujeme chybu na **základě lokálních znalostí**, tak vlastně opravujeme **výskyt chyby, ale ne její příčinu**.
- nelze jej realizovat **několika skupinami současně, paralelně**. Bez plánu vznikají komunikační problémy.

V čem je podstata všech uvedených potíží? V tom, že systémy jsou **příliš složité** a **rozhodnutí jsou prováděna na základě lokálních znalostí**. Neznáme, nevidíme, nejsme schopni mentálně uchopit systém jako celek.

?Role modelování a metodiky je taková, že odstraňuje tyto problémy (asi)

- Systém dělá něco jiného než by měl **- Když by byl model/plán tak se tomu předešlo**
- Systém řeší problémy lokálně. Důsledkem je, že jedna a tatáž věc je řešena na různých místech několikrát, po každé jinak. – **Při existenci modelu by se věc opravila centrálně/globálně**
- Opravy a změny systému jsou velmi obtížné a drahé. (Jestliže opravujeme chybu na základě lokálních znalostí, tak vlastně opravujeme výskyt chyby, ale ne její příčinu.)
- Systém nelze realizovat několika skupinami současně, paralelně. Bez plánu vznikají komunikační problémy.

<u>Metodiky</u>

Chceme-li se vyhnout potížím s lokálním rozhodováním, musíme postupovat **metodicky (ne chaoticky)**, **strukturálně, dle „dobrých“ osvědčených vzorů.**

Návod jak postupovat nám dávají **ověřené postupy – vypracované metodiky.**

Metodiky odrážejí určité náhledy na „realitu“, říkají **„jaké“** kroky učinit v jakém pořadí a **„jak“** je provádět. Dobré metodiky nám říkají i **„proč“** to tak má být.

Metodiky jsou **konservovanou zkušeností** několika generací programátorů a projektantů. Zobecnění principů, zásad, které se osvědčily, viz historie UML.

Modely

Místo abychom se snažili popsat systém jako celek, vytváříme na něj **jednotlivé pohledy** – jeho jednotlivé, dílčí modely. Díváme se na systém postupně z jednotlivých „míst pozorování“, z jednotlivých perspektiv.

Díváme-li se na systém z jednoho místa, opomíjíme vlastnosti z tohoto místa „neviditelné“, nepodstatné a tím si práci zjednodušíme tak, že je mentálně zvládnutelná. **Jednotlivé pohledy jsou jednodušší, zvládnutelné.** Opomíjené vlastnosti se neztratí, jsou hlavními vlastnostmi v jiných pohledech - modelech.

Pohledy musíme volit tak, že postupně popíšeme všechny relevantní vlastnosti systému. Postupně popíšeme vše, co potřebujeme k dosažení stanoveného cíle.

Z jednotlivých pohledů lze zpětně zrekonstruovat celý systém (počítačová tomografie).

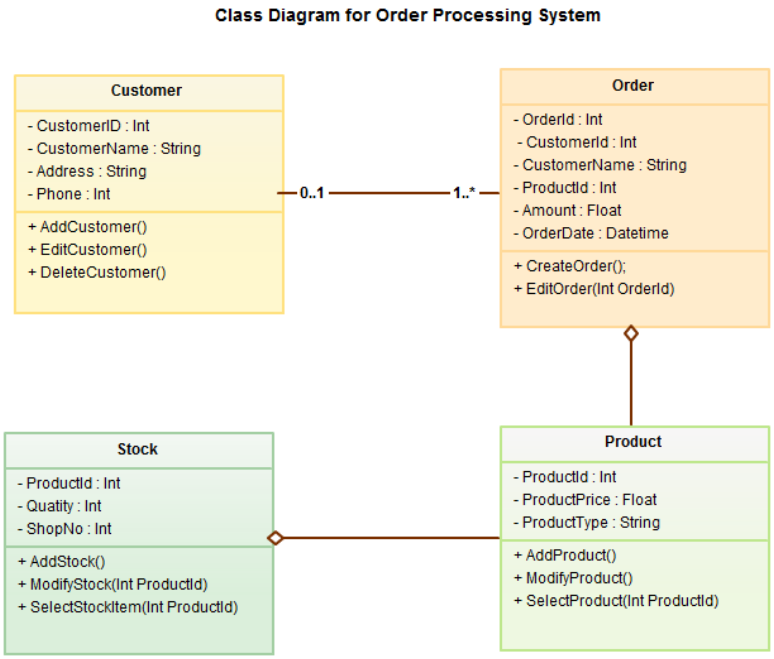
Pro tvorbu různých pohledů jsou obvykle **využity diagramy** – grafické objekty, jejichž kombinací lze tyto pohledy vytvářet.

Diagram je graficky znázorněný model. Diagram popisuje jistou část modelu pomocí grafických symbolů.

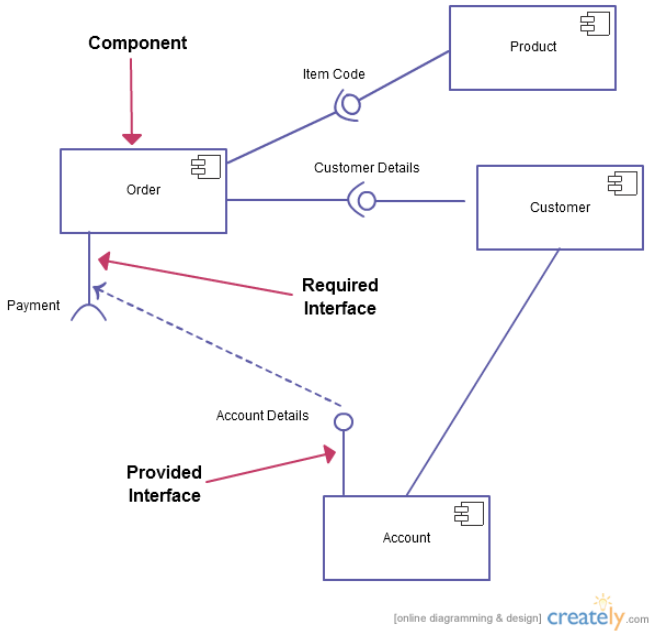
Tento přístup lze přirovnat k modelu stavby, který je tvořen syntézou dílčích stavebních plánů odpovídajících specifickým pohledům na stavbu – plán hrubé stavby, plánu rozvodů elektřiny, plánu rozvodů vody, … V každém z těchto plánů jsou zobrazeny pouze elementy modelu podstatné pro daný pohled, od ostatních elementů modelu je abstrahováno. **Pohledy nejsou nezávislé**, dohromady tvoří konzistentní pohled na systém, tedy konzistentní model.

Pro tvorbu diagramů systému, jejichž syntézou bude model, definuje např. **UML devět typů diagramů.**

1. **Class Diagram**
   Zobrazuje třídy, atributy, operace nad třídami (metody) a vztahy mezi třídami

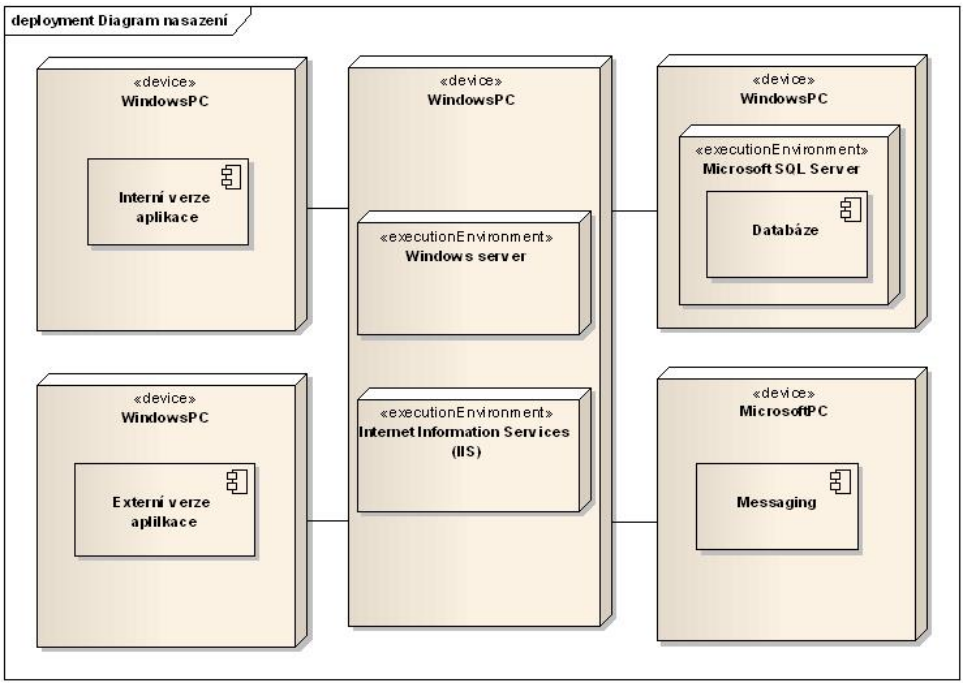


2. **Component Diagram**
Zobrazuje vztahy jednotlivých komponent tvořících aplikací, systém, nebo podnik



3. **Deployment Diagram = diagram nasazení**

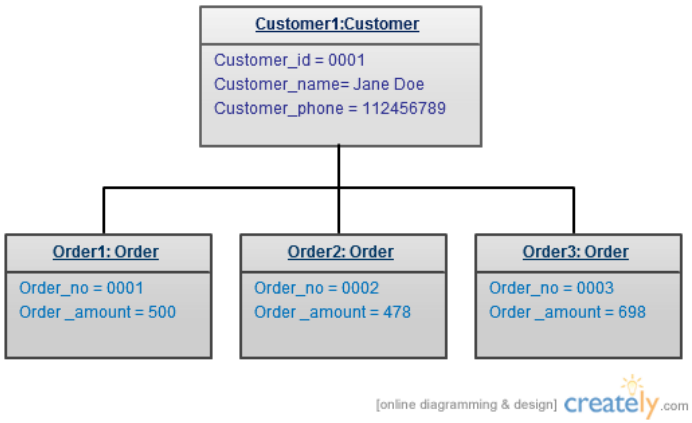
Diagram nasazení (Deployment Diagram) ukazuje „rozložení jednotlivých softwarových komponent na hardwarových zdrojích (uzlech) a jejich spolupráci.”



4. Object Diagram (Instance diagram)

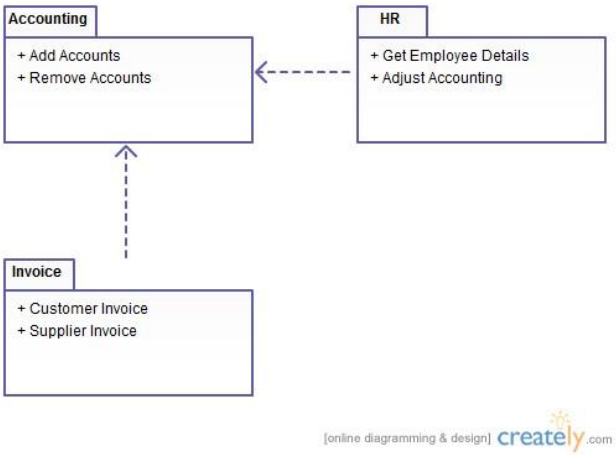
Objektový diagram (Object Diagram) je snímkem objektů a jejich vztahů v systému v určitém časovém okamžiku. [Buch2007] Z důvodu, že zobrazuje instance tříd, je také nazýván diagramem instancí. Používá se především pro znázornění určité konfigurace objektů či zobrazení vzájemně propojených objektů ve speciálních situacích, kdy je diagram tříd či sekvenční diagram nepostačující.

Obvykle obsahuje pouze objekty (objects) a spojení (connections) mezi nimi. Atributy se u objektů vyzna————————————————ují pouze v případě, že je to nutné pro jejich jednoznačnou identifikaci, metody se neuvádějí vůbec.



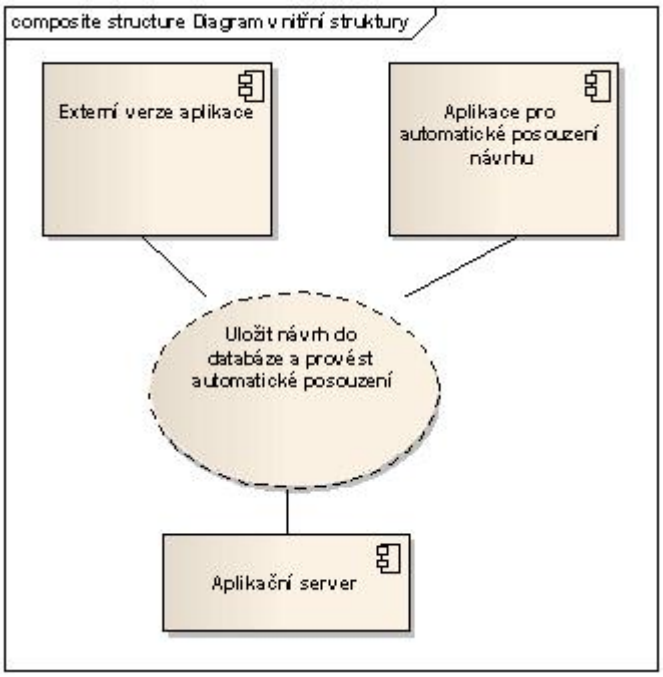
5. Package Diagram

Diagram balíčků (Package Diagram) umožňuje sdružit elementy modelů UML (např. třídy či případy užití) do skupin a mezi těmito skupinami, které se nazývají balíčky, znázornit závislosti



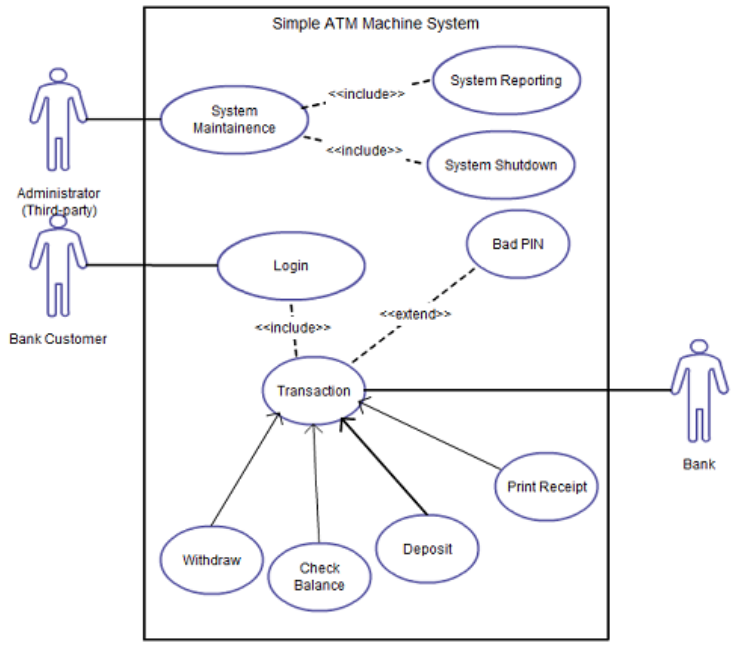
7. **Structure Diagram (popis vnitřní struktury třídy) – diagram vnitřní struktury**

Diagram vnitřní struktury (Composite Structure Diagram) umožňuje znázornit interní strukturu komplexního prvku (třídy, komponenty, případu užití) a zobrazit spolupráci tohoto prvku neboli klasifikátoru s ostatními prvky v systému.



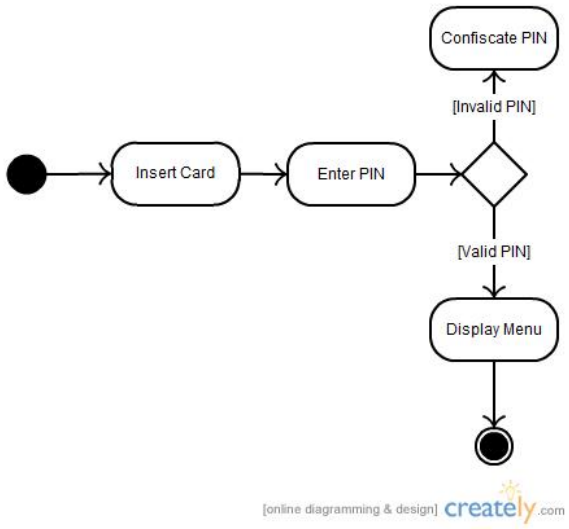
8. **Use Case Diagram – diagram případů užití**

Diagram případů užití (Use Case Diagram) se používá k popisu chování systému z hlediska uživatele a zachycuje, které typy uživatelů se systémem pracují a jaké činnosti v rámci systému vykonávají. [Buch2007] Umožňuje znázornit funkční požadavky na systém tím, že popisuje interakci mezi ním a uživateli



9. Activity Diagram

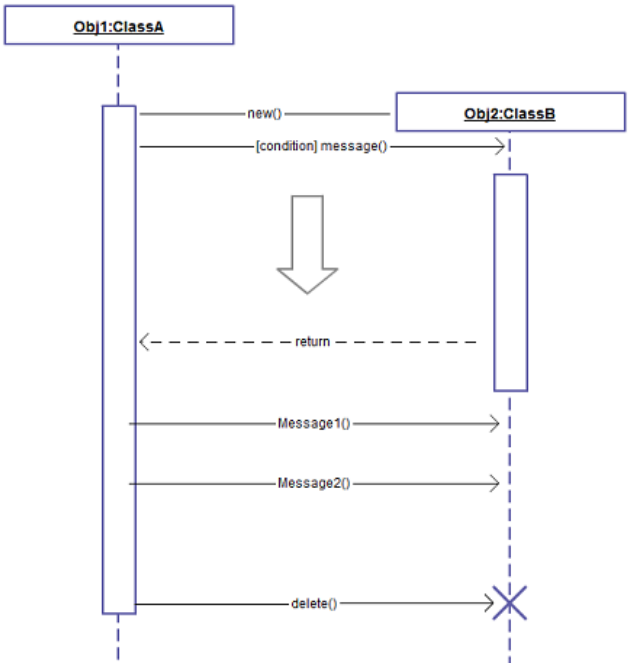
Diagram aktivit (Activity Diagram) je typem diagramu interakcí, který se používá pro popis procedurální logiky, byznys procesů (viz obrázek 17) či pracovních postupů.



11. Sequence Diagram

Sekvenční diagram (Sequence Diagram) je nejvíce používaným diagramem interakcí. *„Zachycuje grafický průběh zpracování v systému v podobě zasílání zpráv.“* [Buch2007]

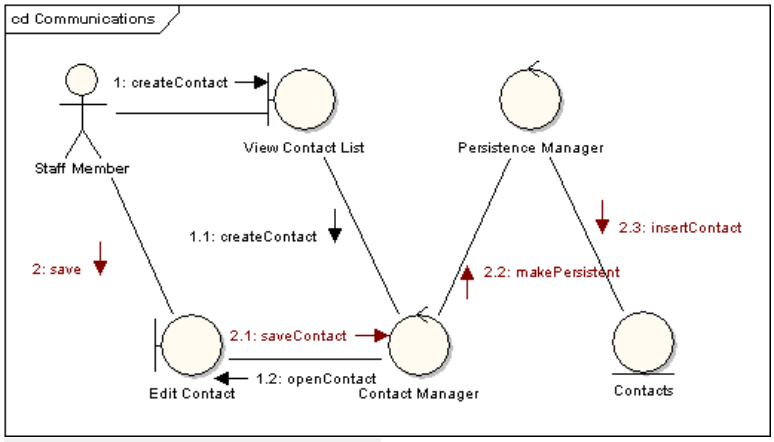
Sekvenční diagram nejčastěji zobrazuje chování a spolupráci jednotlivých objektů v rámci jednoho případu užití. Pro popis chování jednoho objektu napříč více případy užití se používá stavový diagram.



12. **Communication Diagram**

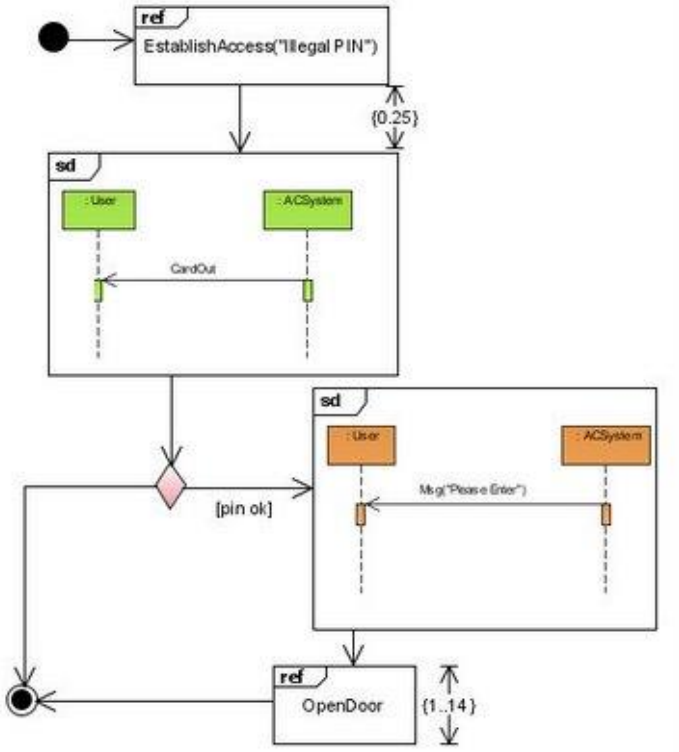
Diagram komunikace (Communication diagram) patří do skupiny diagramů interakcí. „Zachycuje instance tříd, jejich vzájemné vztahy a tok zpráv, které mezi nimi probíhají.“ [Buch2007]

Na rozdíl od sekvenčního diagramu, diagram komunikace klade důraz na strukturální aspekty interakce tím, že umožňuje libovolné rozložení instancí tříd, propojuje je a používá číslování pro znázornění pořadí jejich komunikace



13. **Interaction Overview Diagram**

Přínos tohoto diagramu spočívá v tom, že dokáže srozumitelně a přehledně znázornit větvení i souběžnost a zachytit procesy protínající více případů užití.



3. Metodika návrhu a realizace informačního systému – strukturální a objektová analýza

Existuje v zásadě několik metodik, které popisují analýzu, návrh a realizaci informačních systémů

Jedná se o více méně podrobný popis postupu, který vede návrháře jasně definovanými fázemi krok po kroku při vytváření IS

Metodiky jsou často obecné a každá firma si je může přizpůsobit pro vlastní potřebu a prostředí

Mezi nejpoužívanější patří **Strukturální analýza** (ta je nejstarší), **SSADM** (vznik 80. léta ve Velké Británii) a **Objektová analýza** (konec 80. let, 90. léta)

Metodiky návrhu a realizace informačního systému

Fenomén úhlu pohledu
Chceme-li se vyhnout potížím s lokálním rozhodováním, musíme postupovat metodicky (ne chaoticky), strukturálně, dle „dobrých" osvědčených vzorů.

Návod jak postupovat nám dávají ověřené postupy – vypracované metodiky.

Metodiky odrážejí určité náhledy na „realitu", říkají „jaké" kroky učinit v jakém pořadí a „jak" je provádět. Dobré metodiky nám říkají i „proč" to tak má být.

Metodiky jsou konservovanou zkušeností několika generací programátorů a projektantů. Zobecnění principů, zásad, které se osvědčily, viz historie UML.

Místo abychom se snažili popsat systém jako celek, **vytváříme na něj jednotlivé pohledy – jeho jednotlivé, dílčí modely**. Díváme se na systém postupně z jednotlivých „míst pozorování", z jednotlivých perspektiv.

Díváme-li se na systém z jednoho místa, **opomíjíme** vlastnosti z tohoto místa „neviditelné", nepodstatné a tím si práci zjednodušíme tak, že je mentálně zvládnutelná. Jednotlivé pohledy jsou jednodušší, zvládnutelné.

Opomíjené vlastnosti se neztratí, jsou hlavními vlastnostmi v jiných pohledech - modelech.

Pohledy musíme volit tak, že postupně popíšeme všechny **relevantní vlastnosti systému**. Postupně popíšeme vše, co potřebujeme k dosažení stanoveného cíle.

Z jednotlivých pohledů lze zpětně **zrekonstruovat celý systém** (počítačová tomografie).

Pro tvorbu různých pohledů jsou obvykle využity **diagramy – grafické objekty**, jejichž kombinací lze tyto pohledy vytvářet.

Diagram je graficky znázorněný model. Diagram popisuje jistou část modelu pomocí grafických symbolů.

Tento přístup lze přirovnat k **modelu stavby**, který je tvořen **syntézou dílčích stavebních plánů** odpovídajících specifickým pohledům na stavbu
– plán hrubé stavby, plánu rozvodů elektřiny, plánu rozvodů vody, … V každém z těchto plánů jsou zobrazeny pouze elementy modelu podstatné pro daný pohled, od ostatních elementů modelu je abstrahováno. Pohledy **nejsou nezávislé**, dohromady tvoří **konzistentní pohled na systém**, tedy konzistentní model.

Pro tvorbu modelu systému, respektive pro **tvorbu pohledů na s**ystém, jejichž syntézou bude model, definuje např. UML **devět typů** diagramů.

Zkušenosti ukazují, že je účelné **tvorbu (návrh a realizaci) IS organizovat do posloupnosti etap**, mluvíme o tzv. **životním cyklu IS**.

Životní cyklus IS není statická sekvence činností. Popisuje dynamiku vývoje, měnící se vnější podmínky (změny legislativy), postup od obecného ke speciálnímu. Životní cyklus IS začíná prvotní představou o systému a končí vyřazením systému z provozu. Samozřejmě není znám přesný a úplný (matematický) model životního cyklu. Nepřesné modely ŽC jsou však nepostradatelné pro řízení projektů.

Základní fáze životního cyklu IS:

- Stanovení globálních cílů, specifikace požadavků, specifikace vlastností, které by měl budoucí systém realizovat (implementovat).
- Analýza systému, tvorba analytického, logického modelu systému, model požadovaných vlastností systému.
- Návrh (design), specifikace způsobu, jak požadované vlastnosti implementovat
- Implementace systému, převedení navrženého systému do spustitelného kódu
- Testování vytvořeného kódu
- Nasazení systému, provoz, údržba

V rámci životního cyklu IS řešíme i řadu organizačních a ekonomických otázek:
proč, pro koho, termíny, cena, pracovní tým, situace na trhu, návratnost investice, řízení pracovního týmu, hardwarové prostředky, dokumentace, reakce na změny.

Model vodopád

Pro řízení a vývoj IS by bylo ideální, kdyby po úplném ukončení jedné fáze, etapy ŽC následovala další a k předchozí etapě by nebylo nutné se již vracet.

Model vodopád je složen z posloupnosti vymezených činností.

Realita je však díky své složitosti jiná. Jednotlivé fáze, etapy ŽC se překrývají. Tak jak postupujeme v ŽC, postupně upřesňujeme výchozí poznatky a musíme se vracet k předchozím etapám. V průběhu práce se také mění výchozí požadavky uživatelů a vnější (legislativní, technologické prostředí).

Prototypový model

Tento model se začal prosazovat v 80.letech. Jeho hlavním cílem je urychlení vývoje IS využitím prototypů.

Prototyp můžeme chápat jako zjednodušenou implementaci celého systému nebo jako plnou implementaci části systému.

Prototyp je provedena v co nejkratším čase a v takové funkčnosti, která umožní ověřit, otestovat požadované vlastnosti (např. umožňuje zákazníkovi reagovat na výsledky). Na základě vyhodnocení vlastností prototypu jsou upřesňovány požadavky a modifikován další vývoj.

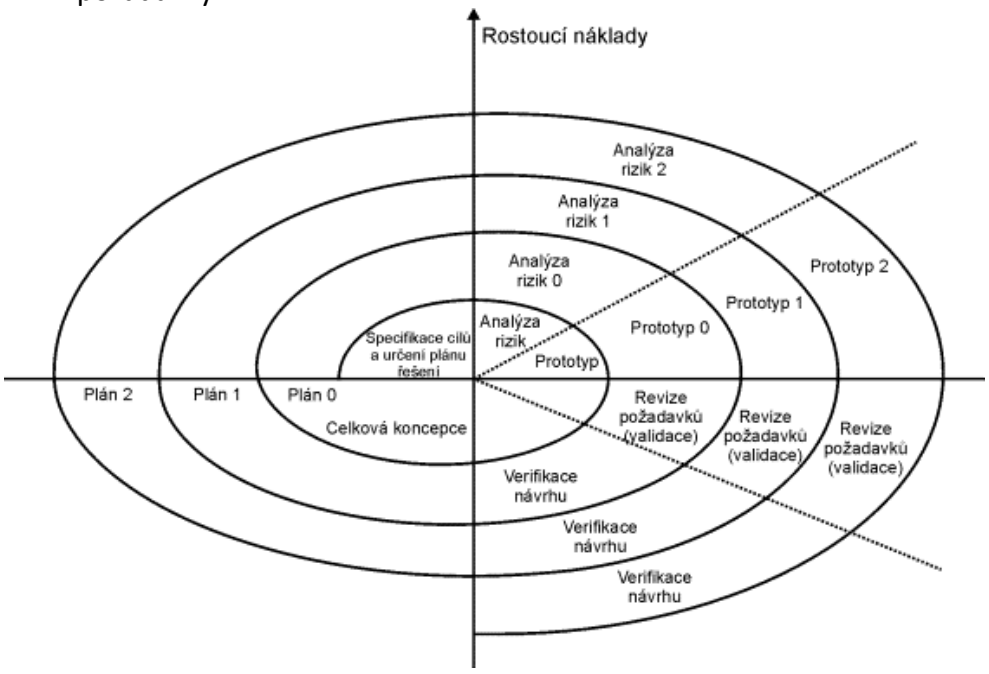
Spirálový model

Tento model vytvořil B.W.Boehm v roce 1988 a je kombinací prototypového přístupu a analýzy rizik.

Základem celého modelu je neustálé **opakování vývojových kroků** tak, že v každém dalším kroku se na již ověřenou část systému přibalují části na vyšší úrovni.

Postup vývoje v jednotlivých krocích se skládá z následujících částí.

- Specifikace cílů a určení plánu řešení.
- Vyhodnocení alternativ řešení a **analýza rizik s daným řešením souvisejících** (je daná alternativa vyhovující, únosná).
- Vývoj prototypu dané úrovně a jeho předvedení a vyhodnocení.
- Revize požadavků neboli validace (testování zda prototyp pracuje tak jak má).
- Verifikace, neboli ověření zda celkový výstup daného kroku je v souladu se zjištěnými požadavky.



Úhlová dimenze modelu reprezentuje postup prací v čase, radiální dimenze pak rostoucí náklady. Nevhodné prototypy jsou opuštěny. Každý nový průchod cyklem rozvíjí nejlepší prototyp.

Nevýhodou modelu je špatný odhad termínu dokončení projektu a jeho celková cena. Proces vývoje je jakoby otevřen.

Strukturální a objektová analýza

Strukturální analýza

Původní strukturální přístup k analýze IS spočíval v **rozdělení** systému na funkční a datovou část (např. DFD diagramy a ER model).

Přínosem byla funkční hierarchická dekompozice – psaní programu shora dolů a datové konceptuální modelování.

Rostoucí složitost systémů (magická hranice 1000 entit a 10000 funkcí) znemožňuje soudržnost datové a funkční vrstvy. Konstruovali se matice, kde řádky jsou datové entity a sloupce funkce systému. Koexistence se označovala křížkem – možnost dekompozice systém – diagonální matice.

Objektová analýza

Výstupem analýzy jsou dva klíčové diagramy:
Diagram analytických tříd a diagramy realizace případů užití

Objektový model – diagram tříd

Objektový přístup čelí kromě jiného složitosti systému tím, že třída - objekt jako nositel (funkční) odpovědnosti – dovednosti, plně **odpovídá** za svá data.

Objekt má svou identitu, vlastnosti, chování a **odpovědnost**. Síla odpovědnosti spočívá v tom, že je **nedělitelná** – žádný jiný objekt nemůže odpovědnost sdílet – dělit se o ni, plést se do ní.

Modelování tříd a objektů je **klíčová aktivita** objektově orientovaného vývoje.

Třída je popisem **množiny objektů** sdílejících stejné vlastnosti - atributy, chování – operace (metody) a vztahy.

Objektový model – diagram tříd
Původní strukturální přístup k analýze IS spočíval v **rozdělení** systému na funkční a datovou část (např. DFD diagramy a ER model).
Přínosem byla funkční hierarchická dekompozice – psaní programu shora dolů a datové konceptuální modelování.

Rostoucí složitost systémů (magická hranice 1000 entit a 10000 funkcí) znemožňuje soudržnost datové a funkční vrstvy. Konstruovali se matice, kde řádky jsou datové entity a sloupce funkce systému. Koexistence se označovala křížkem – možnost dekompozice systém – diagonální matice.

Objektový přístup čelí kromě jiného složitosti systému tím, že třída - objekt jako nositel (funkční) odpovědnosti – dovednosti, plně **odpovídá** za svá data.

Objekt má svou identitu, vlastnosti, chování a **odpovědnost**. Síla odpovědnosti spočívá v tom, že je **nedělitelná** – žádný jiný objekt nemůže odpovědnost sdílet – dělit se o ni, plést se do ní.

Modelování tříd a objektů je **klíčová aktivita** objektově orientovaného vývoje.

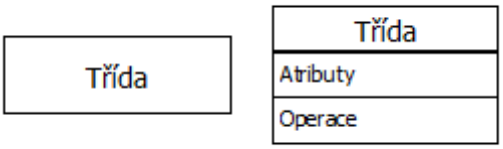
Třída je popisem **množiny objektů** sdílejících stejné vlastnosti - atributy, chování – operace (metody) a vztahy.
Objekt je instancí třídy (chybně se pojem třída a objekt volně zaměňují).

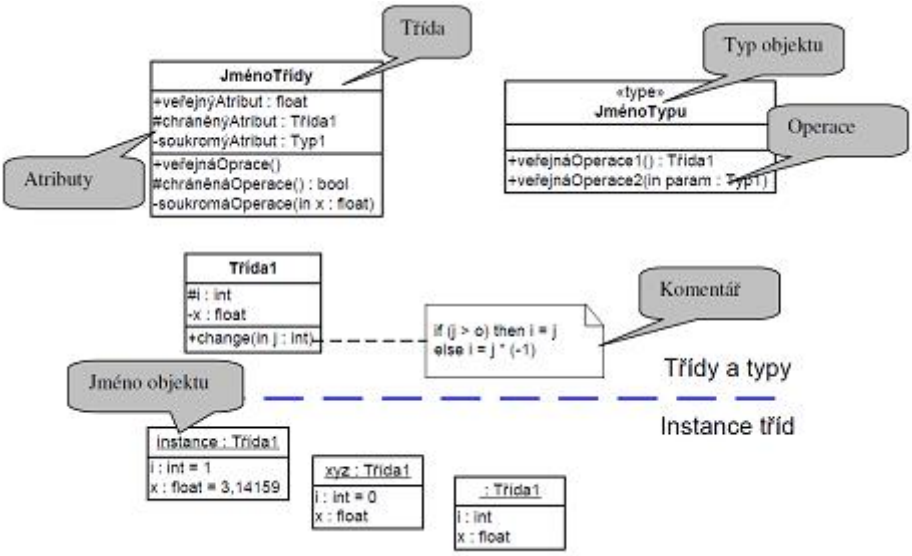
Definice – J. Rumbauh: objekt je diskrétní entita s jasně definovaným rozhraním, které zapouzdřuje stav a chování.

Třídu si můžeme představit jako razítko, objekty jsou pak otisky tohoto razítka, které vidíme na papíře.

Při návrhu třídy neuvažujeme o konkrétním naplnění atributů, pouze určíme jejich název a typ. Teprve při vzniku instance objektu se atributům přiřadí skutečné hodnoty.

Třída je jednoznačně určena svým názvem (v příslušném názvovém prostoru – balíčku). Pro třídu je možno definovat vlastnosti - atributy (Attribute) a chování - operace (Operation). Vizuálním elementem:





Hledání tříd, jejich atributů a kompetencí - vyberme z reality objekty, kandidáty pro zobecnění na třídy a prověřme jejich vhodnosti:

- Potenciální třída je smysluplná, pokud je nezbytná pro funkci systému.
- Potenciální třída je dostatečně stabilní a invariantní vůči vnějším změnám např. technologie, legislativy apod.

Hledání tříd na základě analýzy podstatných jmen a sloves.

Analyzujeme jazyk problémové domény, např. text sebraných požadavků. Podstatná jména a jejich spojení mohou označovat třídy nebo atributy.
Slovesa mohou označovat odpovědnosti, chování tříd.

Pozor na skryté, utajené třídy, které nejsou v textu uvedeny.

4. Datové modelování, perzistence objektů, konceptuální a fyzický datový model

Datové modelování

- jedna ze základních funkcí IS je ukládání a následné zpracování informací - ve formě dat, proto se provádí při návrhu IS také datové modelování
- jeho úkolem je zvolit jaké objekty, jako nositele informace, potřebujeme ukládat do trvalé paměti a jaké jejich vlastnosti a vztahy mezi nimi chceme uchovávat
- při datovém modelování obvykle **vytváříme konceptuální** a **fyzický datový model** a také určujeme způsob perzistence objektů
- Ve fázi návrhu IS je třeba rozhodnout, jak budou objekty, resp. jejich kolekce (třídy), jakožto nositelé informace ukládány do paměti, jak bude probíhat jejich **perzistence.**
- Datové modelování obvykle řadíme do fáze návrhu IS. Musíme zvolit **technologii perzistence**. Jaké objekty, kdy a jak budou ukládány do paměti.
- **Cílem datového modelování je navrhnout „kvalitní" datovou strukturu a databázový systém pro konkrétní IS.**

Technologie obecně

Standardně se uvažuje o technologické koncepci **relačních databází**. Relační databáze jsou na vrcholu svých schopností, jsou to inteligentní „stroje" na ukládání a následný výběr dat ve **víceuživatelském prostředí**. Jsou konstruovány na realizaci velkého počtu současně probíhajících malých transakcí (operací) nad uloženými daty.

Relační databáze jsou v rukách velkých korporací, byly do nich nainstalovány miliardy dolarů, stále běží normalizační proces.

Již delší dobu se hovoří o **objektových databázích**, které by celý proces návrhu a realizace IS zjednodušily. Objekty se svými explicitními vazbami (asociacemi) by se přímo ukládaly do objektové databáze. Proč tomu tak není? (žijeme v epoše dokonalých spalovacích motorů a relačních databází). Musíme perzistentní objekty mapovat, transformovat na relace, tj. tabulky relační databáze.

Perzistence objektů

- zabývá se kam a jakým způsobem budou objekty trvale uloženy
- objekty se obvykle ukládají do relační databáze ve formě datových záznamů v tabulkách
- pro programový přístup do databáze se často používá standardizované softwarové API pro databáze jako ODBC nebo JDBC
- pro usnadnění programátorské práce při vytváření perzistentní vrstvy můžeme využít objektově-relační mapování, které nám zajistí automatickou transformaci ukládaných objektů do záznamů relační databáze - např. framework Hibernate pro Java aplikace

2 druhy datových modelů

Při datovém modelování vytváříme nejprve logický - **konceptuální datový model**. Konceptuální datový model představuje určité zobecnění oproti implementaci datové struktury v konkrétní relační databázi, je nezávislý na konkrétní databázi.

Zvolíme-li konkrétní databázi (např. Oracle) mluvíme o **fyzickém datovém modelu,** na který je konceptuální model převeden.

Z fyzického modelu můžeme generovat SQL skripty, případně se napojit přímo na databázi. Na fyzické úrovni můžeme psát uložené procedury a triggery.

A) Konceptuální datový model

"Konceptuální modelování má své kořeny v počátcích datového modelování a úzce souvisí i například s teorií relačních dat. Jedná se o **modelování reality prostřednictvím základních pojmů (konceptů) a jejich vzájemných souvislostí.** Konceptuální modelování je dnes široce rozvinutý samostatný obor s propracovanými nástroji a metodami, jejichž principy se odráží i v pravidlech modelování tříd objektů pomocí UML."

- **vyjadřuje jaké objekty a jejich atributy budeme ukládat a vztahy mezi nimi**
- **pro grafické vyjádření se používají ERA modely (diagramy)**

Při datovém modelování vytváříme nejprve logický - konceptuální datový model. Konceptuální datový model představuje určité zobecnění oproti implementaci datové struktury v konkrétní relační databázi.

Konceptuální model – fáze návrhu, volby technologie, nezávislý na konkrétní databázi

B) Fyzický datový model

- odvozuje se z konceptuálního modelu a **vyjadřuje navíc jak přesně** budou data v konkrétní databázi **uložena**
- také se popisuje ERA modely, které obsahují i přesné databázové typy atributů tabulek

Zvolíme-li konkrétní databázi (např. Oracle) mluvíme o fyzickém datovém modelu, na který je konceptuální model převeden.

Fyzický model – fáze implementace pro konkrétní databázi, konkrétní implementace. Z fyzického modelu můžeme generovat SQL skripty, případně se napojit přímo na databázi. Na fyzické úrovni můžeme psát uložené procedury a triggery.
Dva přístupy:

- **Při tvorbě konceptuálního datového modelu vycházíme z diagramu analytických, resp. návrhových tříd s jasnou představou o požadavcích na perzistenci. Postup od objektové analýzy, přes návrhové třídy k implementaci.**

Primárně pracujeme s konceptuálním modelem, objektová nástavba je sekundární.

ERA modely

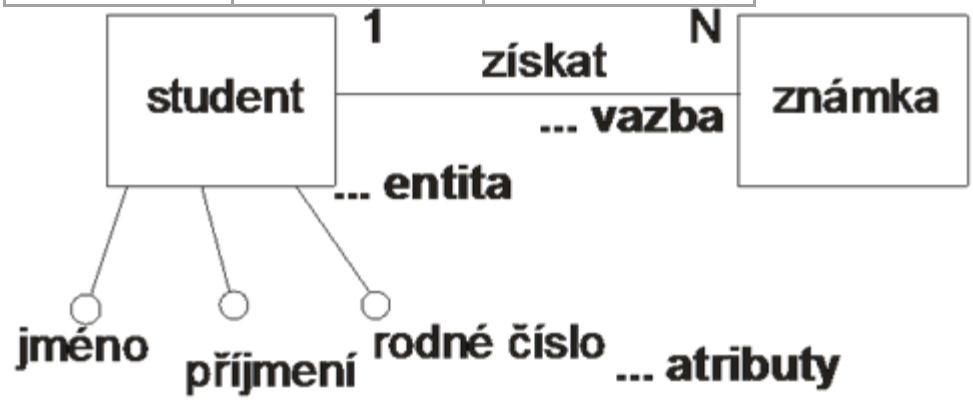
viz DB1 - vypsáno z DB1

Entita – model z reálného světa

Relation – podchycuje vztahy mezi entitami

ERA modely = modelová analýza

Datový model	Datová struktura	Souborový přístup
Entitní množina	Tabulka	Soubor
Entita	Řádka	Záznam
Atribut	Název sloupce	Položka



Vazby

1:N - vyjadřuje, že jedné entitě E1 může příslušet více entit z entitní množiny E2

jedné entitě z E2 přísluší jen jedna entita z E1

nejlépe 1:N(0)

př.: student – známka

1:1 - na vazbě se podílí jen jedna entita z E1 a jedna z E2

vždy se ptát, proč je to rozdělené, proč to není jedna entita

vazba je zajímavá, pokud alespoň jeden konec volný (nepovinný)

př.: student – známka (student nemusí mít známku)

N:N - jedné entitě z E1 přísluší více entit z E2

jedné entitě z E2 přísluší více entit z E1

př.: student – rozvrhová akce

ER modely

- primární – minimální množina atributů, která jednoznačně určuje entitu
- na vazbu se díváme jako na entitu – lze k ní přidat atributy (čtenář – výpůjčka – exemplář)
- vazba 1:N se při realizaci vytvoří tak, že do "podřízené" tabulky přenesu klíč (cizí klíč) z "nadřazené" tabulky
- vazba M:N nelze realizovat, nelze vyřešit pomocí cizích klíčů = musí se provést rozklad vazby
- mezi dvěma entitními množinami může existovat více vazeb
- vazba nemusí být binární, ale může být n-ární
- vazba může být i unární (sama na sebe)

Některé datové modely rozlišují entitní množiny regulární a slabé

Slabá entitní množina je taková, u níž nelze určit, či nemůžeme zjistit nadřazenou množinu

FAN

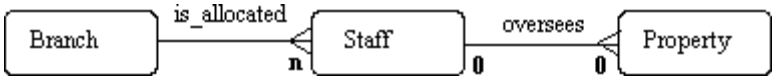
Problém: 2 vazby 1:N se větví z jedné entity

datové položky ve dvou složkách nejsou v přímém vztahu, ale mají vazbu založenou na datových položkách ve třetí složce

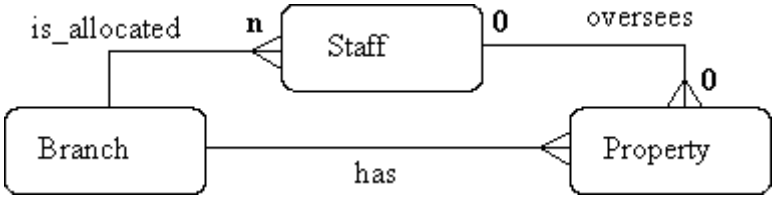
Řešení:

CHASM trap

Problém: Existuje vztah mezi entitami, ale chybí vazba.



Řešení:



Objektově relační mapování - Perzistence

Objektově relační mapování – O/R slouží k tomu, aby bylo možné snadno používat relační databáze v prostředí objektově orientovaných programovacích jazyků.
Vzhledem k tomu, že objektově orientovaný návrh dat není jednoznačně převoditelný na relační databáze a opačně, používají se různé formy mapování.
Mapování má za účel načítat data z relační databáze a naplnit jimi příslušné datové položky objektů včetně vazeb mezi objekty, případně naopak datové položky objektů ukládat do databáze.

Snahou ORM je co nejlepší využití obou zmíněných technologií

- objekty by měly reprezentovat objekty reálného světa, jak to požadují principy OOP
- na straně databáze bychom zase měli využít všech možností relačních databází – indexy, pohledy, primární klíče, triggery a uložené procedury.

Alternativou k O/R mapování je použití objektové databáze, která je navržená přímo pro ukládání objektů. Použití takové databáze eliminuje potřebu převádět data z objektové podoby do relační. **Data jsou uložena přímo ve své objektové reprezentaci.**
Objektové databáze zatím nejsou příliš rozšířené.
V současnosti je nejpoužívanějším nástrojem pro ORM produkt od firmy JBOSS **Hibernate.**
Hibernate je O/R mapovací nástroj pro jazyk Java. Jde o volně šiřitelný open source software. Nabízí prostředí pro mapování objektového modelu na tradiční relační schéma.

Perzistentní třídy musí splňovat jisté vlastnosti, obsahovat

- konstruktor bez parametrů
- getter a setter metody pro perzistentní položky

Konstruktor bez parametrů Hibernate používá při načítání objektu z databáze. Jeho zavoláním v paměti vytvoří prázdný objekt, jehož položky následně nastaví podle hodnot uložených v databázi pomocí setter metod. Getter metody Hibernate používá při čtení položek při ukládání objektu do databáze.

Základní mapování - mapování tříd na tabulky
Perzistentní (entitní, bussines) třídy, resp. jejich instance - objekty odpovídají entitám konceptuálního datového modelu, resp. řádkům tabulek fyzického modelu. Atributy třídy se stanou sloupci tabulek.

Mapovací soubory pro Hibernate se píší v jazyce XML nebo se využívá tzv. anotací Javy (zápis metadat přímo do kódu). Každá třída se mapuje na jednu tabulku. Každá tabulka musí obsahovat primární klíč. Mapovací soubor obsahuje výčtem elementů <property>, které reprezentují položky, které mají být ukládány a jak mají být ukládány.

Způsob uložení položek lze ovlivnit velkým množstvím atributů. Několik nejpoužívanějších popisuje následující výčet.

- column="column_name" - určuje název sloupce. Implicitně se používá název proměnné.
- type="typename" - určuje typ konverze mezi Java typem a SQL typem.
- not-null="true|false" určuje zda je možné do daného sloupce uložit NULL hodnotu.

Mapování atributů musí odpovídat konverzi datových typů, norma SQL – 92 definuje standardy datových typů (opakem jsou transientní třídy).

Mapování vztahu

Mapováním vztahů zajišťujeme, že se může mezi objektovým modelem a databází současně „přenášet“ síť vzájemně provázaných – asociovaných objektů. Základní výhoda ORM
Při použití Hibernatu jako ORM vrstvy se vazby dělí ještě na jednosměrné a obousměrné.
Rozdíl je v tom, že u jednosměrné vazby má referenci jen jedna entita. Druhá tedy žádnou referenci nemá, kdežto u obousměrné vazby mají reference obě entity.
Vztah se v mapovacím souboru mapuje pomocí jednoho z elementů

- <one-to-one>
- <one-to-many>
- <many-to-one>

- <many-to-many>

podle kardinality daného vztahu. Jediným povinným atributem je name, ostatní atributy jsou nepovinné. Nepovinné atributy jsou podobné jako u elementu <property>.

Mapování dědičnosti

Protože cílový fyzický datový model nepřipouští dědičnost tabulek, viz norma SQL 92, existují tři možnosti:

- mapování 1:1 – každá třída se mapuje do samostatné tabulky, jedna instance objektu je rozložena po více tabulkách, řada nevýhod.
- zahrnutí do nadtřídy – atributy podtříd jsou zahrnuty do nadtřídy, z třídy a jejich podtříd vnikne jedna tabulka. Vhodné v případě malého počtu podtříd.
- rozpuštění do podtříd – všechny atributy nadtřídy jsou přeneseny do tabulek pro všechny podtřídy. Počet tabulek odpovídá počtu podtříd. Vhodné pro velký počet podtříd.
  Viz CASE

5. **OLAP systémy, jejich význam a oblasi využití, základními prinicpy, dimenze, agregace, extrakce a transformace dat, srovnání transakčních a analyzických system (OLAP a OLTP technologií**

6. Vlastnosti a typy CASE nástrojů a jejich význam v analyze a návrhu onformačních systémů

CASE - obecně
CASE – Computer Aided Software Engineering – jsou programy určené k tomu, aby **podporovaly vývoj informačních systémů**.

Společný pohled - používání CASE nástrojů umožňuje designerům, programátorům, testerům a manažerům (tedy všem, kteří se na vývoji systému podílejí) mít **společný náhled** na to, jak projekt vypadá jako celek, jak jeho jednotlivé části v detailu a jaký je jeho stav v jednotlivých fázích vývoje.

CASE:
- **pomáhá** zajistit to, že proces vývoje projektu je řízený, řiditelný a kontrolovatelný.
- **čelí složitosti** systému, která by bez jejich pomoci byla těžko zvládnutelná.
- **zajišťuje kvalitu procesů** vývoje softwaru (díky použitým metodikám a odhalování chyb při jejich použití)**.**
- **zajišťuje** značnou **úsporu času** (a tedy nákladů) potřebného k vývoji systému.
- **slouží jako úložiště** projektové dokumentace.

Některé CASE nástroje jsou přímo integrovány do vývojových prostředí.

Odhady hovoří o tom, že použití CASE (přes počáteční zpomalení) nástrojů představuje úspory okolo **50 až 70 procent** v dalších etapách životního cyklu softwaru.

CASE nástroje jsou založeny na dvouvrstvé architektuře.
Základ každého z nich tvoří tzv. „**repository“**, kam se ukládají veškeré informace o navrhovaném systému (jedná se o databázi, která automaticky udržuje data v konzistentním stavu).
Nad společným repository pracuje **druhá modelová vrstva**, která zpřístupňuje informace uložené v repository.

Každá z modelových vrstev se opírá o jistou metodiku a reprezentuje jistý pohled na informace uložené ve společném repozitury. Jednotlivé modely jsou díky společnému repozitury na sebe vzájemně převoditelné (např. z diagramu tříd můžeme vygenerovat fyzický datový model).

Vývoj a typy CASE nástrojů

- CASE systémy vznikly v **sedmdesátých letech dvacátého století**, v situaci, kdy začala prudce narůstat složitost IS.

- CASE se na trhu začaly výrazně prosazovat zhruba *v* **polovině osmdesátých let**.
- Tyto systémy vznikly v okamžiku dosažení kritické kvality v metodách, organizaci práce a technologiích, potřebných při vývoji informačních systémů. **Od podpory čistě vývojových fází** životního cyklu IS (analýzy a konstrukce systému) k **podpoře strategických rozhodování** na počátku projektu a operativních činností souvisejících s provozem systému a řízením jeho změn a rozvoje.

- To, co dalo podnět ke vzniku CASE nástrojů a určuje také směry dalšího vývoje, je **metodikou tvorby informačních systémů – strukturální a objektové metodiky.**

- Postupem doby a s měnícími se požadavky dnes existuje celá řada CASE nástrojů. Je to díky podporovaným metodikám a samozřejmě také tím, v **jaké fázi vývoje je nástroj používán.**

- **Cílem je využít CASE ve všech fázích životního cyklu IS od s**pecifikace požadavků, analýzu, návrh a kódování po údržbu IS.

Kategorie CASE nástrojů

Podle podporovaných fází životního cyklu systému lze CASE nástroje rozdělit do dvou základních kategorií:

- **Integrované CASE**. Zaměřují se na podporu celého životního cyklu vývoje IS.

- **Specializované CASE**. Tyto nástroje jsou orientované na určité specifické etapy.

Specializované CASE nástroje

Nástroje používané v různých etapách se liší. Většinou pokrývají jen určité činnosti. Podle životního cyklu vývoje lze CASE nástroje rozdělit dle na:

- ***Pre CASE*** –
  Tyto nástroje jsou určeny pro tvorbu celkové strategie IS.

- ***Upper CASE*** –
  Nástroje této kategorie podporují plánování, specifikaci požadavků, modelování organizace podniku a celkovou analýzu IS**.**

  Hlavním úkolem je analýza organizace, zachycení všech procesů, definice klíčových toků a dokumentace zjištěných požadavků. Použití je pro specifikaci cílů a počátečních požadavků a řízení projektu.

- ***Middle CASE*** –
  Tento druh CASE nástrojů je základem všech komerčně dodávaných nástrojů.

  Prostředky této kategorie slouží pro podrobnou specifikaci požadavků analýzu a návrh systému.
  Používají se také pro dokumentaci a vizualizaci systému.

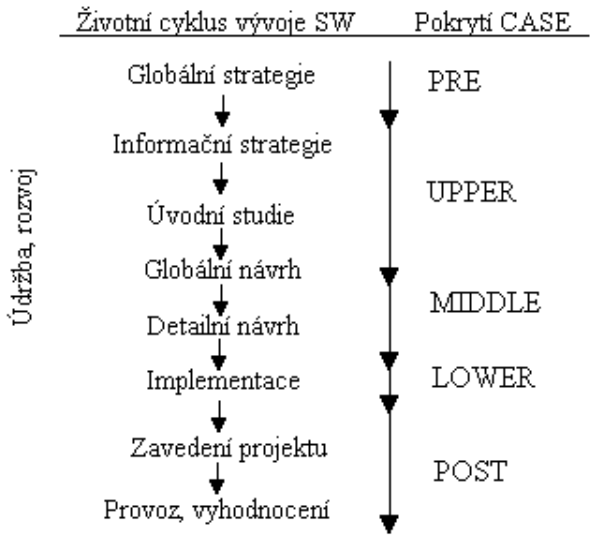
- ***Lower CASE*** –
  Tyto nástroje slouží především jako podpora kódování, testování a údržby. Jejich součástí jsou i **nástroje forward engineeringu**[1], tedy generátory kódu z modelu (ty mohou generovat kostru nebo podstatnou část výsledného kódu, programátor poté doplňuje jen nutné detaily a algoritmy). Dále pak jde o prostředky pro **reverse engineering**[2], které umožňují získat model z již existující aplikace, prostředky pro plánování a zjišťování kvality SW (sběr informací o testování, vyhodnocení testů, řízení testování), pro správu konfigurace, prostředky pro sledování a vyhodnocování práce systému. Funkce CASE nástrojů této kategorie se často překrývají s funkcemi obecných vývojových prostředí.

- ***Post CASE –***
  Tento druh CASE nástrojů podporuje organizační činnosti jako zavedení, údržbu a rozvoj IS.

Působnost takto rozdělených CASE nástrojů se překrývá, protože jimi podporované činnosti se mohou vyskytovat v různých fázích životního cyklu vývoje IS.

Fáze životního cyklu IS a CASE nástroje

Vztah CASE nástrojů a fází životního cyklu IS je zachycen na následujícím obrázku:



Pravdivé a mylné představy o CASE nástrojích

Pravdivé představy:

- Hlavním přínosem těchto nástrojů je vytváření úplných podkladů pro programování aplikací.
- CASE jsou nástroje, které mohou zlepšit produktivitu práce, efektivita práce vždy závisí na osobních kvalitách jednotlivých pracovníků.
  - Mohou generovat části kódu, ale nenahrazují programovací jazyky.
  - Praxe ukázala, že CASE nástroje často selhávají právě díky nedisciplinovanosti uživatelů.
- Automatizací „chaosu“ vznikne automatizovaný „chaos“.
  - Na počátku práce je nutné vykonat velmi mnoho činností, jejichž výsledek není dlouho vidět.
  - Dostanou-li stejný CASE dva systémoví analytici, dospějí k dvěma naprosto odlišným řešením.

Mylné představy:

- CASE nástroje slouží jako náhrada programovacích jazyků.
- Všechny CASE nástroje pracují podobně (poskytují stejné výstupy).
- Užívání CASE nástrojů zlepší práci manažerů organizace využívající výsledný produkt.
- CASE odstraňuje potřebu disciplíny a přísného vývoje aplikací IT.
- Od CASE nástrojů se často očekává jako výstup tvorba aplikačního programového vybavení.
- Produktivita dosažená pomocí CASE je okamžitě zřejmá.
- Užívání CASE zaručí konzistenci výstupů.