Analýza informačních systémů

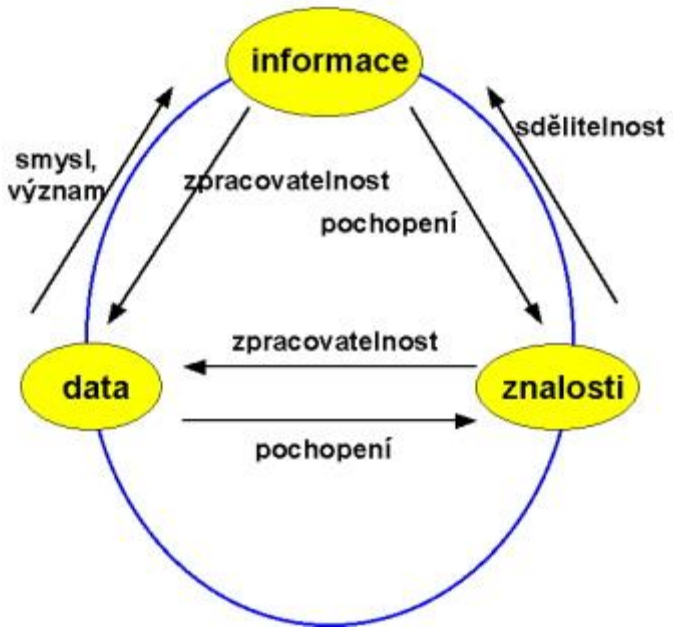
Informační systém (IS) je systém pro sběr, udržování, zpracování a poskytování informací.

Pojem informace patří k nejobecnějším kategoriím současné vědy, filozofie i umění (hmota, vědomí, …).

Podle toho, v kterém vědním oboru nebo, v které oblasti lidské činnosti se používá, jsou aplikovány specifické přístupy ke zkoumání informace (kybernetika, sociologie, ekonomika …).

Informace - data - znalosti
Data -jakékoli vyjádření (reprezentace) skutečnosti, schopné přenosu, uchování, interpretace či zpracování. Umožňují přenášet a zpracovávat odraz skutečnosti.
Znalosti - to, co jednotlivec vlastní (ví) po osvojení dat a informací a po jejich začlenění do souvislostí. Výsledek poznávacího procesu, předpoklad uvědomělé činnosti. Účel znalostí - porozumět skutečnosti.

Informace definovaná pomocí dat a znalostí:
data, která mají smysl (význam) sdělitelné (komunikovatelné) znalosti



Informační technologie - nástroje, prostředky a techniky, které si člověk vyvinul v oblasti výpočetní techniky, telekomunikací a zpracování informací.
Informační technologie - technologický potenciál, který svými důsledky a možnostmi začíná zasahovat do mnoha oborů lidské činnosti. V této souvislosti hovoříme o **informační společnosti.**

Informační technologie jsou a vždy budou nástrojem, který lidé mohou používat k tomu, aby lépe a efektivněji vykonávali to, co považují za potřebné či vhodné.

Informační technologie radikálně mění naše dosavadní představy o tom, co je dosažitelné, realizovatelné - možné.

Automatizovaný informační systém
Informační systém založené na informačních a komunikačních technologiích.

Vzhledem k rychlému pronikání informačních a komunikačních technologií do společnosti přívlastek Automatizovaný ztrácí smysl.

Cíl tvorby (automatizovaného) informačního systému:

definování struktury digitálně uložených dat a vytvoření programu, který bude realizovat požadované činnosti. Zdroj: Encyclopaedia Britannica (heslo Information processing)

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)
- uložení informací (zaznamenávání, shromažďování na nosiči)
- přenos informací
- zpřístupnění informací (tisk, zobrazení, šíření…)

Současné typy (a terminologie) 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

- APS – advanced planning and scheduling: systémy na podporu vnitropodnikového (dílenského) plánování,

- SCM – supply chain management:plánování dodavatelských logistických řetězců
- HR – human resources – řízení lidských zdrojů

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

- MIS – management information system,
- EIS –executive information system,
- DSS – decision support system,
- 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

Systémy pro tvorbu a správu dokumentů

- DTP – desktop publishing
- 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.

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

Další členění informačních systémů

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

Computer aided technologie (CAD, CAM, CIM, CASE...)
Podstata: počítačová podpora (automatizace) některých procesů (návrh, výroba ap.)
CAD (*computer-aided design) počítačem podporované projektování.* Jde o velkou oblast IT, která zastřešuje širokou činnost navrhování.
Ve strojírenství CAM (computer-aided manufacturing) CAE (computer-aided engineering)

Stavebnictví a architektura – AEC (Architecture-Engineering-Construction), BIM (Buliding Information Model), CAAD (Computer-aided architectural design)

Role modelování a metodiky pří tvorbě IS

Fenomém složitosti
Se složitostí světa roste i **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.

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 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 **několikrát, 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.

Mluvíme-li o systému, rozumíme tím vždy soubor komponent (prvků) a vztahy (vazby) mezi nimi.

Termín "složitost" se vztahuje ke studiu složitých systémů, jejichž přesná definice však dosud neexistuje (sociální, ekonomické, biologické systémy). Známe však řadu projevů, vlastností složitých systémů.

Zhruba řešeno, systém označíme za složitý, pokud se skládá z řady navzájem interagujících komponent (komponent ve vzájemných vztazích) a objevují se u něj vlastnosti (chování), které není samozřejmým důsledkem vlastností (chování) jednotlivých komponent. Ze znalosti části, z lokálních znalostí nelze jednoduše „složit“ znalost celku. Složitý systém nás vždy překvapí.

Ze složitosti IS vycházejí tzv. empirické „zákony“ vývoje software

„Zákon“ invariantní spotřeby práce: Rychlost vývoje systému je v podstatě konstantní a nekoreluje s vynaloženými prostředky.
Zákon omezené velikosti přírůstku: Systém určuje přípustnou velikost přírůstku v nových verzích. Pokud je limita překročena, objeví se závažné problémy.
Zákon trvalé proměny: Systém používaný v reálném prostředí se neustále mění, dokud není cenově výhodnější systém nahradit zcela novou verzí.

Zákon rostoucí složitosti: Při změnách je systém (program) stále méně strukturovaný a vzrůstá jeho složitost. Odstranění narůstající složitosti vyžaduje dodatečné (neplánované) zdroje.

Metodiky návrhu informačních systémů
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.

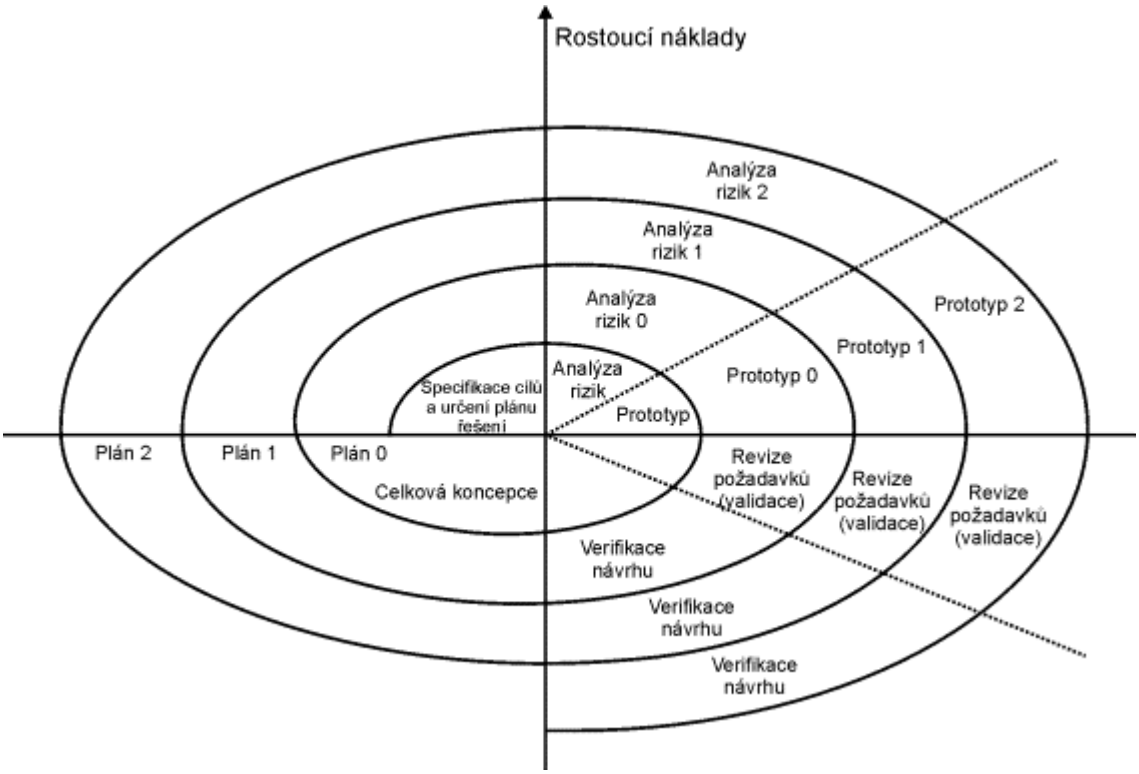
Prototyp je provedena v co nejkratším čase a v takové funkčnosti, která umožní ověřit, otestovat požadované vlastnosti (např. umožňuje zákazníkovi reagovat na výsledky). Na základě vyhodnocení vlastností prototypu jsou upřesňovány požadavky a modifikován další vývoj.

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.

Pokrytí životního cyklu IS
Dnes existují desítky metodik, které **pokrývají** jednotlivé fáze životního cyklu IS.

Je zřejmé, že žádná metodika není a ani nemůže být v praxi striktně vynucována a dodržována ve všech svých detailech.

Metodika je určitou kostrou poskytující základní rámec pro vývoj systému, poskytuje návod jak postupovat v jednotlivých fázích životního cyklu informačního systému.

Všechny části metodiky mohou být a pravděpodobně budou uživateli upravovány podle specifických potřeb. Firma volí své firemní metodiky.

Metodiky jsou obsahově naplňovány jednotlivými
- **metodami**, s nimi souvisejícími
- **technikami** a k tomu potřebnými
- **nástroji**

Výše uvedené pojmy bývají často zaměňovány.

Základní principy metod analýzy:
- princip abstrakce:
- princip modelování

Abstrakce znamená:
Myšlenkový proces odlučující odlišnosti a zvláštnosti a zjišťující obecné a podstatné (společné) vlastnosti předmětů a jevů skutečnosti a vztahů mezi nimi.

Motivace - **snaha po rozdělení zkoumané problematiky na mentálně zvládnutelné části**. Odloučit se, „abstrahovat“ od detailů.

Vytváření abstraktních pojmů sdružováním prvky:

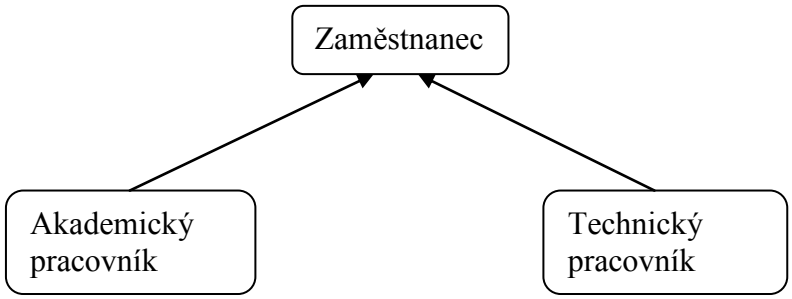
- **Agregace (kolektivizace)**, kde abstraktní pojem vyjadřuje pouze účelové sdružení svých prvků a nedefinuje jejich společné vlastnosti.
  Členění celek - část
  Příkladem takové abstrakce může být pojem "zpracování účetních dokladů", který zahrnuje řadu procesů, jako "zpracování faktur", "zpracování příjmových dokladů" atd., aniž by přitom všem svým prvkům definoval povinné společné vlastnosti.

- **Generalizace,** kde abstraktní pojem vyjadřuje sdružení prvků na základě jejich společných vlastností, které jsou všemi podřízenými prvky povinně dány.
  Členěné obecné - speciální
  Příkladem takové abstrakce může být pojem "zaměstnanec", zahrnující celou řadu podpojmů, jako "dělník", "manažer", "úředník".

Hierarchická (stromová) abstrakce znamená dělení celku na části, top-down (v opačném směru, sdružování částí do celku). Hierarchická abstrakce definuje stromovou strukturu prvků, pojmů.

Význam hierarchie je dán způsobem dělení celku či slučování částí. Hierarchická abstrakce patří k základnímu vzoru, paradigmatu analytického myšlení.

Abstrakce generalizace, **specifický typ - generický typ** je typickou hierarchickou abstrakcí v datovém a objektovém modelu, kde umožňuje jednotlivé entity (objekty) zobecňovat do vyšších abstraktních celků – generických typů. Vše společné přesouváme na nadřazenou úroveň.



Top-Down hierarchie funkcí a procesů
Top-Down princip je tradičním principem, používaným ke zjednodušení pohledu na popisovaný systém.

Tento princip byl použit v metodách strukturovaného programování a také v metodách analýzy systému, založených na tzv. funkčním přístupu. Je také primárním (dominantním) principem strukturalizace procesů v současných metodách procesní analýzy.

Princip Top-Down spočívá v rozdělení pohledů na zkoumaný systém **podle úrovně podrobnosti pohledu**, kde primárně postupujeme od shora dolů, od obecného ke speciálnímu.

Příklad funkční hierarchie, dělení systému na podsystémy.

Použití tohoto principu je však problematické. Až teprve při zkoumání detailů jsme totiž schopni konkrétně rozhodovat o uspořádání celého systému (tedy i abstraktních - nadřazených funkcích). V praktickém postupu jsme tak nuceni se velmi často vracet a opravovat chybné návrhy vazeb.

Princip modelování
Modelem vždy rozumíme ***abstraktní*** *obraz reality (reálného světa). Formální vyjádření zkoumaného jevu (systému) sloužící jako vyjádření skutečnosti.*

Funkční přístup chápe smysl modelu reálného světa v tom, že obsahuje *souhrn stavů reálného světa a změn těchto stavů*. Funkční přístup je dynamický.

Ke změnám stavů v modelu dochází prostřednictvím *operací* (to je abstraktní obraz událostí). Operace jsou podle různých hledisek a principu sdružovány do vyšších celků - funkcí.

Datový přístup se zaměřuje na *vlastnosti (atributy)* reálného světa, jejichž abstraktním obrazem v IS jsou *data*. Datový přístup je statický.

Datovým modelem reálného světa je potom systém entit, objektů, charakterizovaných jejich atributy, a jejich vzájemnými vztahy. Smyslem modelování z hlediska datového přístupu je především formulovat ideální (konceptuální) podobu uspořádání dat v informačním systému, která je „věrným" obrazem reálného světa.

Princip tří architektur
Princip tří architektur (P3A) definuje způsob použití abstrakce pro vývoj IS po jednotlivých vrstvách (vrstvená abstrakce) dle:

- obsahu
- technologie
- implementace.

Návrh IS potom podle P3A probíhá ve třech po sobě následujících architekturách, modelech, viz životní cyklus IS:

- **Analýza systému, model reality -** obsahový – formální (logický) model systému, nezatížený ani technologickou koncepcí řešení, ani jeho implementačními specifiky.

  Objektový přístup, UML: Diagram tříd, Stavový diagram a sadu doplňkových diagramů (Use Case, Diagram komunikace objektů a Sekvenční diagram).
  Strukturované metody: Diagram datových toků (DFD.

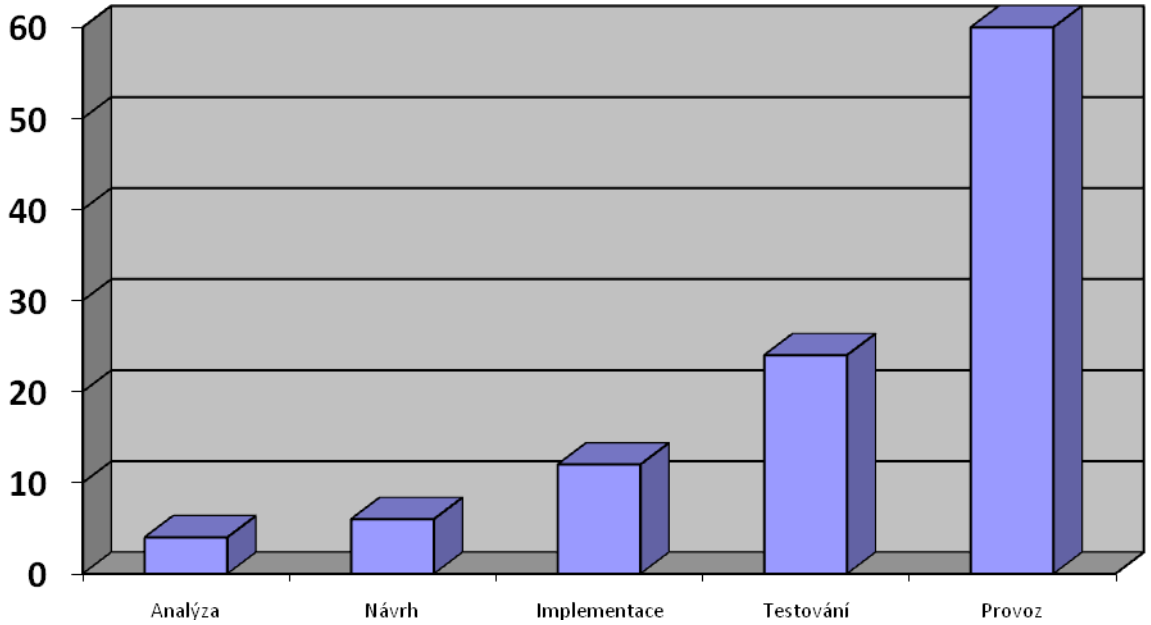
- **Technologický model, návrh systému (design) -** zohledňující technologickou koncepci řešení (např. relační databáze, architektura client – server, konceptuální datový model (ERD)).

- Technologický model stále nesmí být zatížen implementačními specifiky řešení. Technologický návrh určuje, **jak** bude obsah systému v dané technologii realizován.

  Objektový přístup, UML: Diagram komponent a Diagram rozmístění zdrojů
  Strukturované metody: Fyzický datový model.

- **Implementační model** - zohledňující implementační specifika použitého vývojového prostředí (konkrétní databázový systému, programovací jazyky a další implementační prostředky, aplikační server, vývojového prostředí GUI, atd.).

Rozdělení návrhu a realizace do tří oddělených vrstev sebou přináší řadu výhod a pružnost rozvoje IS – výměna technologie jednotlivých vrstev.



Náklady na odstranění chyby v závislosti na fázi životního cyklu IS

Konceptuální a procesní modelování

Základem moderních analytických modelů je **model reality**, sestávající ze dvou, vzájemně souvisejících modelů: **procesního** a **objektového** (dříve strukturální a objektový přístup).

Objektový – konceptuální model představuje statický model reality (businessu) - popisuje z čeho je realita složena a jaké jsou základní, podstatné, tedy *stálé* (*statické*) složky, čili objekty a vazby mezi nimi.
Akce (metody) a algoritmy, vázané k objektům v jejich životních cyklech, zde mají význam též statický, resp. jsou podřízeny tomuto - statickému - pohledu.

Procesní model je modelem dynamickým, popisuje změny - následnosti akcí, vedoucí od počátečních ke koncovým stavům.

Modelování procesů a příklon k procesnímu řízení je jedním z aktuálních trendů současnosti.

Oba modely jsou různými pohledy z různých úhlů na totéž - na realitu, její strukturu a chování.
Každý z obou modelů popisuje realitu z hlediska své dimense – **bytí, versus chování**.

Událostmi motivované procesy změn v realitě se odehrávají v prostoru (ve struktuře, síti) objektů.

Model (entitních, busssines, analytických) objektů, tedy podstaty - struktury reality sleduje základní stavební kameny, z nichž se realita – problémová doména skládá.

K takovému popisu existuje specifický diagram – **Diagram tříd** (Class Diagram - základní diagramem jazyka UML).

Pomocí dalších diagramů jazyka UML, např. **Stavového diagramu** lze model tříd doplnit o specifikaci průběhu tzv. „*životního cyklu*“ objektů.

Model věcných procesů (tedy chování) reality sleduje **řazení akcí** v realitě do jednotlivých (business - podnikových) procesů.
K takovému popisu je zapotřebí tzv. **Procesní diagram**, model, který není standardizován, není součástí jazyka UML.

Požadavky

Požadavkem se obecně rozumí **jednotka potřebnosti funkcionality** nebo jiné vlastnosti systému.

Požadavek lze definovat jako specifikaci toho, co by mělo být implementováno.

Požadavek je popis jisté funkce nebo vlastnosti, kterou by měl budoucí systém realizovat (implementovat).

Inženýrství požadavků (requirements engineering) je termín popisující aktivity zjišťování, dokumentování a údržbu množiny požadavků na systém. Odhalování způsobu jak a k čemu uživatelé daný systém potřebují.

Existují studie, které dokazují, že neúspěch v procesu inženýrství požadavků je hlavní příčinou konečného neúspěch celého systému.

Doporučuje se klást velký důraz na sběr, definici a řízení požadavků (reguirements engineering). Vhodný výběr a zapojení všech skupin budoucích uživatelů.

Pojem požadavek může mít širší, nebo užší význam:

- uživatelský (užší) – znamená subjektivní požadavek, kladený uživatelem na systém,

- obsahový (širší) – znamená požadavek na obsah systému, obecný důvod potřeby jisté funkcionality či vlastnosti systému.

Zdroje požadavků:
- legislativa
- představy uživatelů
- pracovní procesy uživatelů
- Know-how dané problémové oblasti
- hardwarové a softwarové vybavení zákazníka

Dva typy požadavků:
- funkční – specifikují požadavky na funkčnost systému – viz. případy užití, funkcionalita z vnějšího pohledu.
- nefunkční – specifikují další vlastnosti systému, technologické, bezpečnostní, designové (vzhled, ovládání UI).

Požadavek říká **Co** bude systém nabízet, dělat, nikoliv **Jak** to bude uděláno, implementováno. Právě **Co** je společné téma s uživatelem.

Funkční požadavky se spojí – mapují na případy užití, nefunkční se promítají do technické specifikace.
V knihách o UML se říká, že případy užití „zachycují“ funkční požadavky.

Příklad: Knihovna
Zmapováním všech podstatných vlastností:
Uvažujme jednoduchý knihovní systém se čtenáři a knihami. Nebudeme řešit nákup ani případné vyřazení knih. Nebudeme řešit ani registraci čtenáře.
Knihovna má své čtenáře a knihy. Pro každou knihu, pro každý titul může být v knihovně uložen jeden nebo více exemplářů – kopií. Budeme řešit systém půjčování, rezervací a vracení knih.

Primární sběr požadavků
Jako první provedeme sběr funkčních požadavků:

Title ID Text	*Short Description*	*Priority*
1.	Zapůjčení knihy - čtenář si v katalogu může vybrat a posléze zapůjčit knihu - konkrétní exemplář knihy.	1
2.	Rezervace - v případě, že požadovaná kniha není k dispozici, čili všechny exempláře knihy jsou zapůjčené, může si čtenář udělat na knihu rezervaci.	1
3.	Omezení zápůjčky - knihy se zapůjčují na omezenou dobu jednoho měsíce.	1
4.	Vrácení knihy - v okamžiku, kdy čtenář vrací exemplář knihy, zkontroluje systém, jestli čtenář dodržel výpůjční lhůtu. V případě, že lhůta byla překročena, je za každý den prodlení čtenáři naúčtována pokuta x Kč. '	1
5.	Upomínka - má-li čtenář půjčenou knihu déle než jeden měsíc, je mu poslána mailem urgence na vrácení knihy.	1
6.	Výzva při rezervaci - v okamžiku, kdy bude do knihovny vrácen exemplář knihy, na kterou má čtenář rezervace, bude mu poslán e-mail, že si může rezervovanou knihu vypůjčit. '	1

Viz. Power Designer

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.

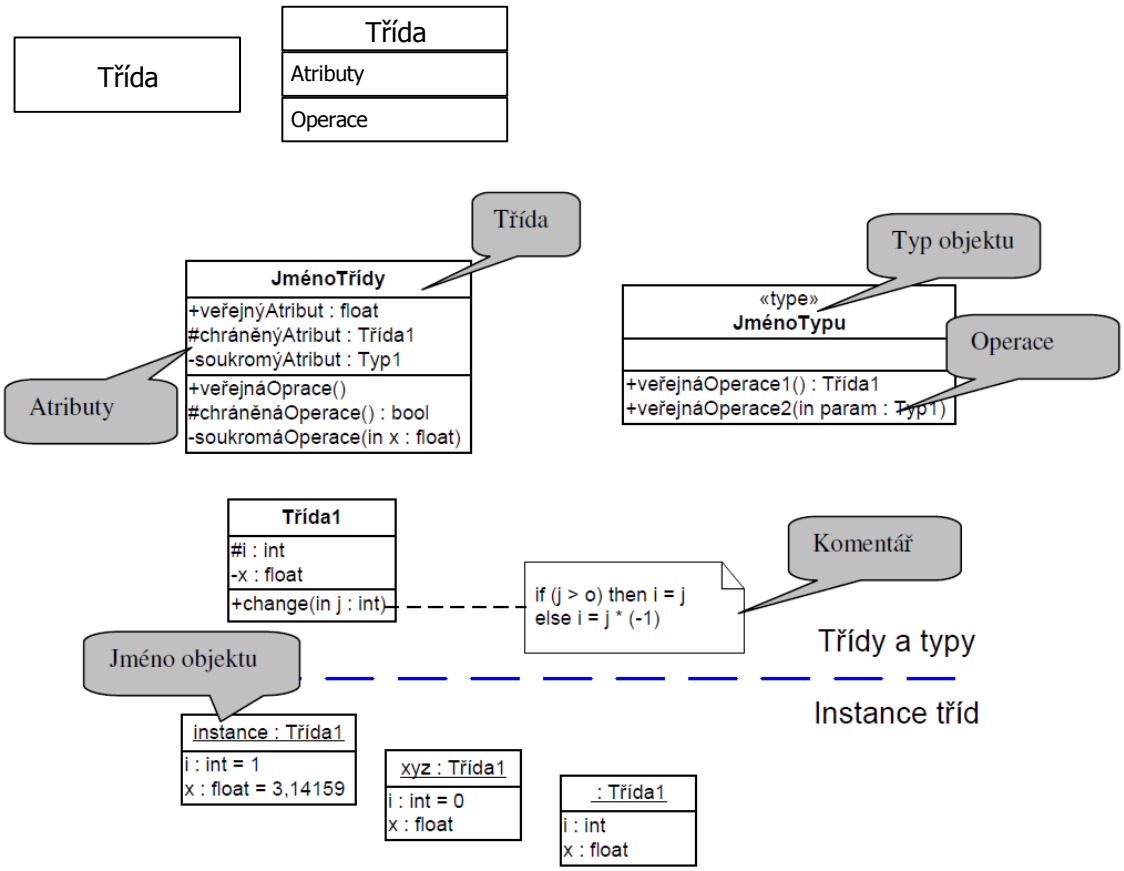
Objekt je instancí třídy (chybně se pojem třída a objekt volně zaměňují).

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.

Příklad: Knihovna

Čtenář – reálný objekt, pan Novák
Knih – z katalogu knihovny – abstraktní objekt, zápis
Exemplář knihy – reálný objekt, konkrétní výtisk dané knihy
Údaje o jedné knize jsou společné pro její výtisky, exempláře, kopie.

Vypůjčují se exempláře knihy, rezervace se dělají na knihy. U zápůjčky nás zajímá datum, kdy se má kniha vrátit zpět. U rezervace nás zajímá datum do kdy je rezervace platná.

Viz. Power Designer

Třída *Čtenář* zachycuje informace o všech lidech - čtenářích registrovaných v knihovně. Soustřeďuje informace potřebné proto, aby si člověk mohl (pouze) zapůjčit a vrátit knihu, aby se stal čtenářem. Zachycuje přípustné chování čtenáře vůči knihovně. Samozřejmě také jeho přihlášení do knihovny (vznik), změny a odhlášení (zánik).

Třída *Čtenář* **klasifikuje** – „značkuje“ lidi z hlediska knihovny. Je deskriptorem množiny objektů daných vlastnosti v dané problémové oblasti.

Klasifikace, zařazení do třídy přenáší **význam** na formální objekt, je nositelem sémantiky. Klasifikace je jedním z nejdůležitějších způsobů, jímž lidé uspořádávají, vnímají, chápou okolní svět.
Existuje mnoho způsobů jak klasifikovat okolní svět, proto je analýza tak náročná.

Definice atributů
Atribut určuje vlastnosti objektu, je nositelem informace o objektu.

Atributy popisují hodnoty (stavy) udržované v jednotlivých objektech.

Objekty jsou vymezeny (popsány) množinou atributů.

Atributy popisují vlastnosti objektů, které potřebujeme k dosažení daného cíle.
Reálný objekt ve své nekonečné složitosti nelze vymezit omezenou množinou atributů.
Základní problém analýzy IS - výběr rozumného množství **relevantních** atributů.

S atributy mohou manipulovat výhradně služby daného objektu.

Klíčovou otázkou je zodpovědnost určitého objektu za uchování informací. Hledání atributů je řízeno otázkami:

- Jak je objekt popsán v kontextu zodpovědností daného systému.
- V jakých stavech se může objekt v průběhu svého životního cyklu nacházet

Specifikace atributů:
Atribut je definován:

- jménem,
- typem (formátem)
- viditelností (veřejný – public, soukromý – private a chráněný – protected).

Každý atribut pečlivě pojmenujte. Volte názvy, které jsou běžné v aplikační oblasti a jsou rozumné délky a pevné struktury. Ke každému atributu připojte vysvětlující text.

Hledejte omezující podmínky pro hodnoty atributů. Omezení se vztahují na:

- formáty, rozsah, výčet přípustných hodnot, přesnost
- implicitní hodnoty, požadavek na nastavení výchozích hodnoty atributu
- prevalidační a postvalidační podmínky – podmínky, které musí být splněny před a po změně hodnoty atributu, za jakých podmínek je povolen přístup k atributu (např. v závislosti na hodnotách ostatních atributů)
- závislost atributů, viz datové modelování, jak změna jednoho atributu ovlivňuje hodnoty jiných – závislých atributů, viz normalizace relačního modelu.

Identita objektu
Objekt je vedle svého stavu a chování jednoznačně určen, **je jedinečný**, má identitu, atribut, který jej jednoznačně identifikuje mezi všemi ostatními objekty dané třídy, má své ID. Atribut zajišťující identitu, se v datovém modelování nazývá primární klíč.

Čtenář (číslo čtenáře, jméno, adresa, kontakt)
Kniha (isbn, autor, titul)
Exemplář (číslo exemp, datum nákupu)
Exempláře knihy se liší inventárním číslem a sledujeme u nich datum nákupu.

Problematika volby primárního klíče, viz dále.
Viz. Power Designer

Umístění atributů
Ve třídách, které jsou vázány dědičností (generalizace) umístíme atribut do co nejvyšší třídy, ve které atribut platí pro všechny její generické podtřídy (specializace).

Hledání tříd, doporučení:

- Každá třída by měla mít 3-5 klíčových odpovědností
- První extrém - není dobré, když existuje velké množství malých tříd
- Druhý extrém – není dobré mít velké třídy
- Nezavádějte „funktiody“ – pro jednotlivé funkce systému nezavádějte třídy
- Vyhýbejte se stromům dědičnosti s mnoha úrovněmi

Vazby – relace mezi třídami

Vazba asociace (Association)

Vazba *asociace* mezi třídami je vyjádřením abstraktního vztahu mezi objekty (instancemi tříd). Asociace říká, že objekty mají mezi sebou přímý vztah, **že o sobě ví.**

Zaměstnanec pracuje v daném oddělení, mohu se ptát: V jakém oddělení pracuje zaměstnanec, mohu získat seznam všech zaměstnanců v oddělení. Vazba je nositelem významu – sémantiky, odpovídá – mapuje požadavky kladené na systém. Je trvalejšího charakteru.

Asociace je společný typ vazby pro:

agregaci, vyjadřující vztah mezi celkem a částí

prostou asociaci, vyjadřující prostou objektovou referenci.

Vazba asociace je specifikována řadou vlastností, z nichž některé jsou vázány přímo k vazbě asociace (například název), ostatní k zakončením vazby (například role). Podrobné určení vlastností až v okamžiku návrhu – specifikace návrhových tříd a jejich vazeb má „implementační“ důsledky.

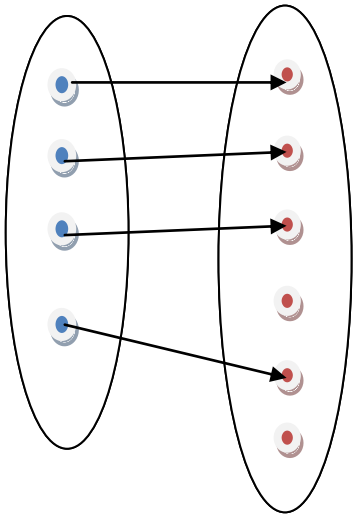
Vazbu asociace lze zavézt jako **orientovanou** (Navigability), přičemž neorientovaná vazba je považována za obousměrnou (dva jednosměrné vztahy).

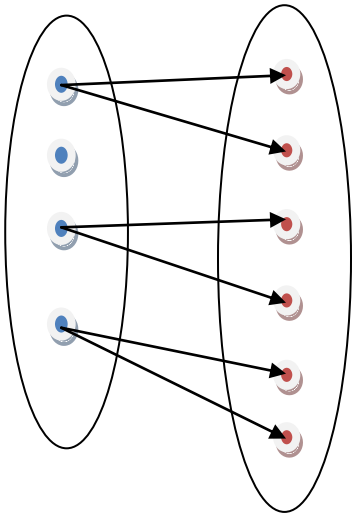
Třídy v asociaci mohou vůči sobě vystupovat v **rolích** (Role) (Objednávka – Zaměstnanec, Zaměstnanec vystupuje ve vztahu k objednávce v roli Prodejce). Každá strana asociace má své jméno – roli. Role popisuje vlastnost, funkci třídy „viděné“ z druhé strany.

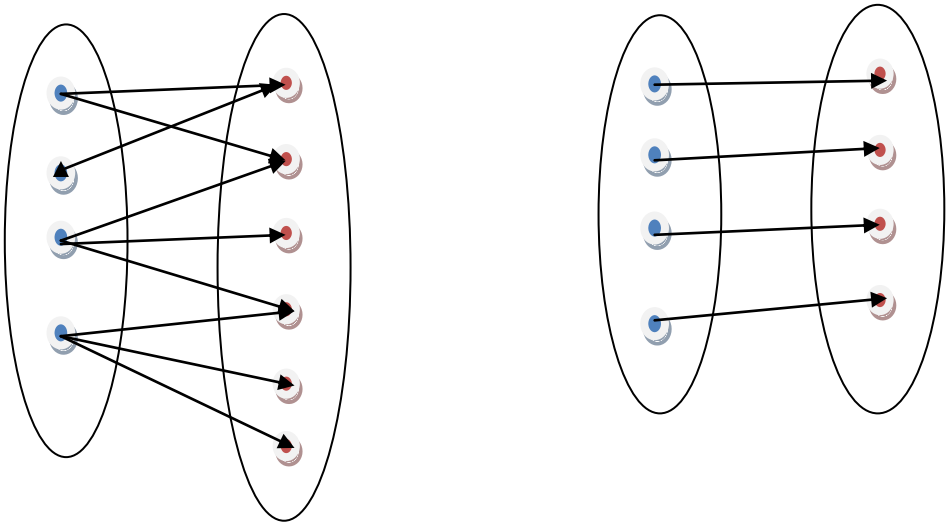
V asociaci lze určit **násobnost vazby, (**kardinalitu) multiplicitu, která vyjadřuje počet možných vazeb objektů tříd v asociaci (0, 0..1, 0..*, 1, 1..*, *, M..N, …).

Násobnost vazby definuje, ***kolik může k jednomu objektu, tj. k jedné instanci třídy A na jedné straně vztahu existovat*** **minimálně** (parcialita) **a maximálně** (kardinalita) objektů ze třídy B na druhé straně vztahu a obráceně.

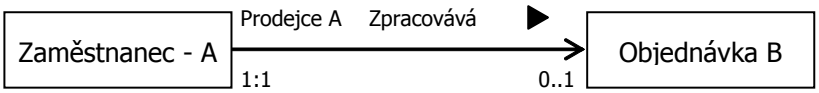
Násobnost (multiplicity) ukazuje počet hodnot (kardinalit), jichž může nabývat příslušná role ve vztahu (role označuje úlohu, kterou má objekt viděný objektem z druhé strany vztahu)







Vizuální element:



Standardně je vazba asociace implementována zavedením atributu třídy - role pro zachycení objektové reference (množiny referencí pro parcialitu 0..*) na objekty druhé třídy. Implementace vazby – objekt si sebou nese reference na asociované objekty.

S vazbami je třeba šetřit.

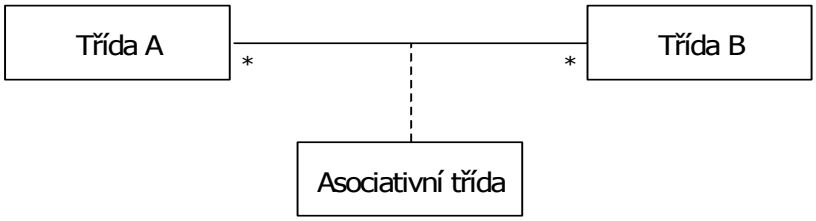
Na rozdíl implementace v relačním datovém modelu se jedná o explicitní vyjádření vazby. V relačním modelu se pro vyjádření vazby používají tzv. cizí klíče – implicitní implementace vazby.

Další vlastnosti vazeb související s implementací vazby (mimo UML, CASE):

Každý konec asociace, vazby se nazývá role. Pro roli můžeme definovat řadu vlastností.

Asociativní třída (Association Class) je vazba asociace, která je rozšířena přiřazením třídy pro zachycení informací nutných pro úplnou specifikaci této vazby.

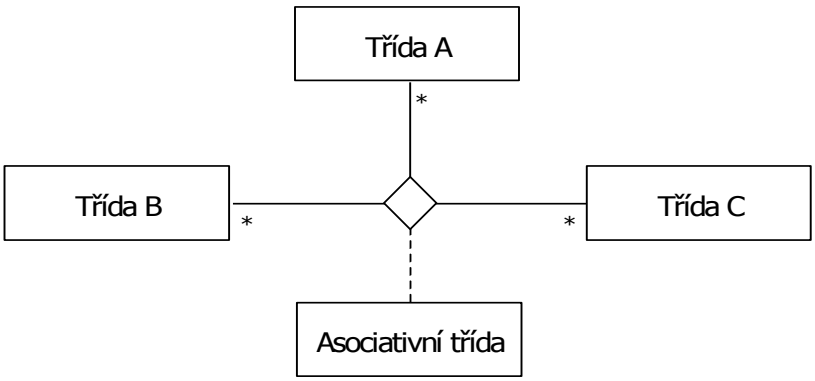
Asociativní třída se používá například v případě oboustranně násobné vazby N:M. Asociativní třída nemá vlastní identitu, identitu přejímá od „asociovaných“



tříd.

Doposud uvedené příklady vazeb asociace popisovaly vždy relaci dvou prvků, jednalo se o *binární asociace* (Binary Association).

Vztah mezi třemi a více prvky popisují *vícenásobné asociace* (N-ary Association). Pro vyjádření vícenásobné asociace se používá element modelu asociativní třída.



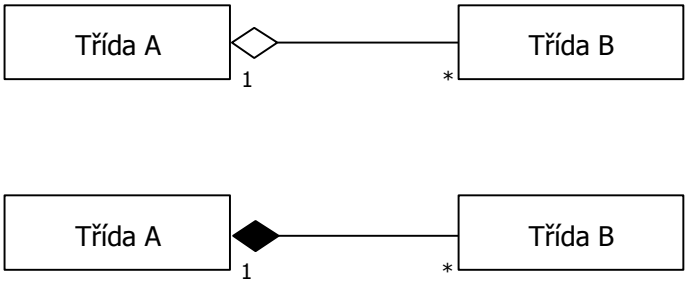
Vazba agregace (Aggregation)
Agregace je vyjádřením abstrakce vztahu mezi objekty (instancemi tříd), který odpovídá vztahu **celku a části** .

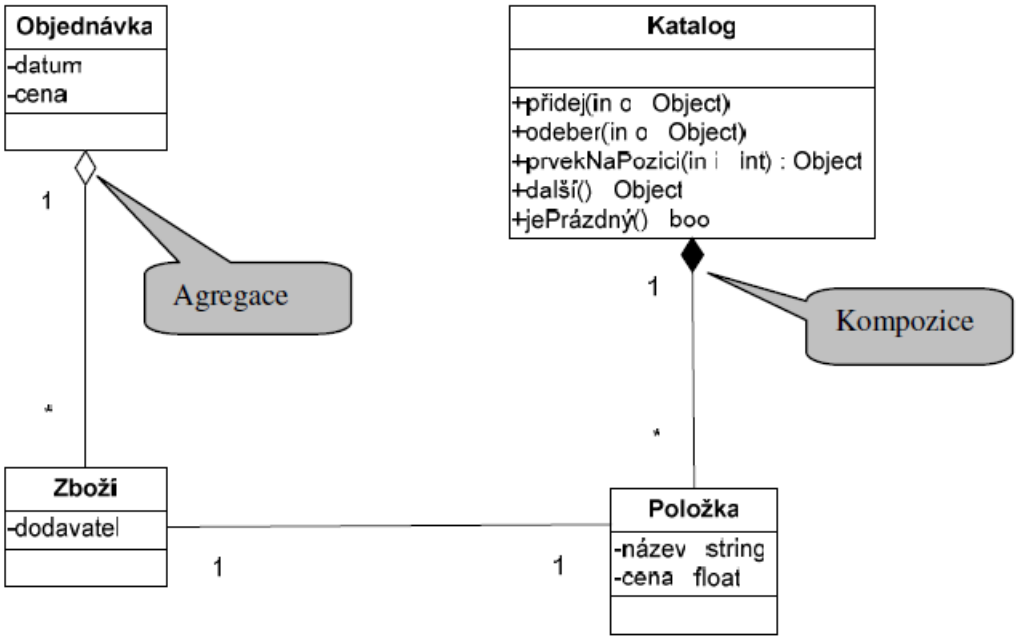
Agregace je speciálním případem asociace.
Jazyk UML rozlišuje mezi dvěma typy agregací:

- prostou agregací (Simple Aggregation)
- kompozicí (Composition).

Vazba kompozice je silnější než vazba prosté agregace, požaduje **vlastnictví** objektu agregované třídy (část) a vyjadřuje **kompetenci** objektu agregující třídy A (celek) k vytvoření a zrušení objektů agregovaných tříd B (část). Třída A vlastní třídu B.

Jedná o vlastnictví částí celkem. Kompozice má důležitou vlastnost z hlediska životního cyklu celku a jeho částí. Existence obou je totiž totožná. Zánik celku (kompozitu) vede i k zániku jeho částí na rozdíl od prosté agregace, kde části mohou přežívat dále jako součásti jiných celku.

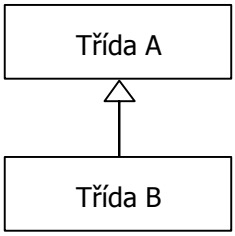


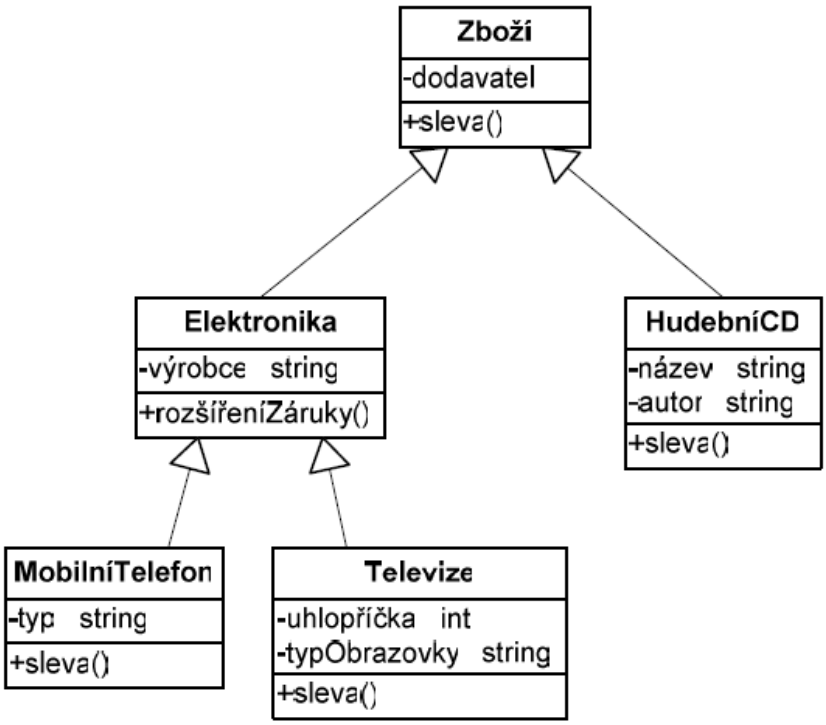


Vazba generaliace (Generalization)
Vazba generalizace je vyjádřením vztahu mezi obecným elementem (Parent) a specifickým elementem (Child), který je konzistentní s obecným elementem a přidává k jeho definici další informace, je tedy bližší specifikací (specializací) obecného elementu.

Vazba generalizace mezi třídami je vyjádřením vlastnosti dědičnosti, jedné ze základních vlastností objektově orientovaného přístupu.

Jazyk UML povoluje vyjádřit vícenásobnou dědičnost zavedením více vazeb generalizace od.





Vazba závislosti (Dependency)
Vazba závislosti umožňuje znázornit jistou závislost mezi elementy modelu.

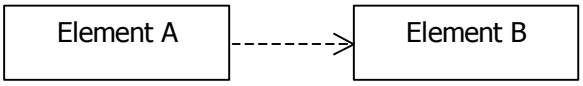
Vazba závislosti je určena svým názvem a obvykle se používá s určitým **stereotypem**, který blíže specifikuje formu závislosti, zavádí její typ.

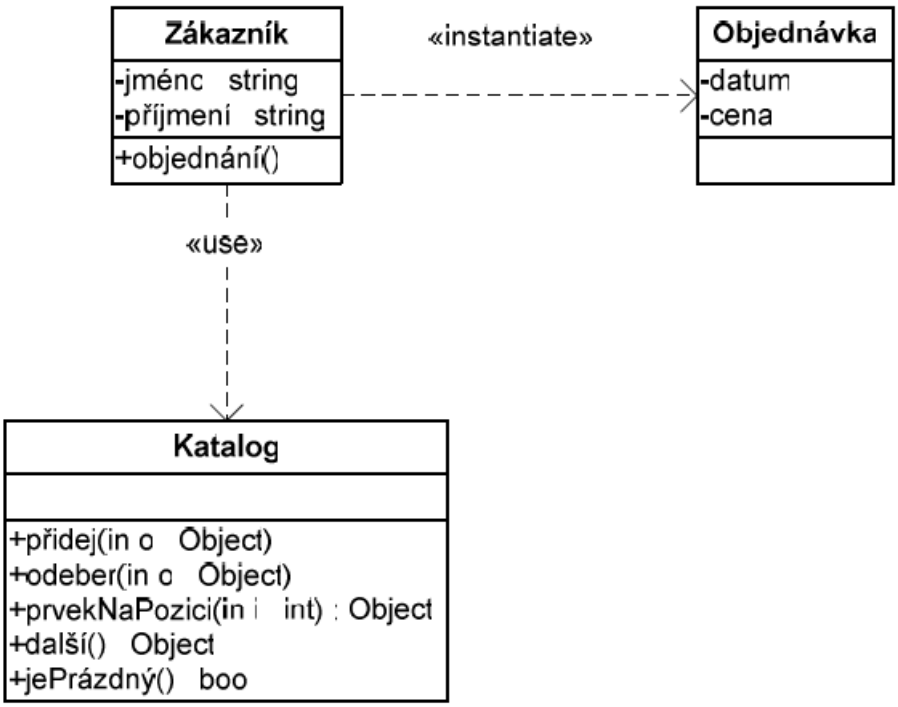
Vazba závislosti je znázorněna orientovanou přerušovanou čárou, kde orientace je vyjádřena šipkou ve směru závislosti.

Závislost obvykle vzniká pouze dočasně pro potřeby poskytnutí služby klientskému objektu a poté tato vazba zaniká (implementační rozdíl od asociace).

Změna jednoho (nezávislého) elementu ovlivní druhý (závislý) element

(Zákazník *využívá instance třídy* Katalog *(stereotyp* use*).*





Zákazník využívá instance třídy Objednávka k vytvoření její instance (stereotyp instantiate). K tomu, aby ji mohl naplnit, dále používá instance třídy Katalog (stereotyp use). V obou případech obě vazby vznikají jen na dobu nezbytně nutnou k poskytnutí požadovaných služeb.

Chování objektů, předávání zpráv
Objekt poskytuje služby prostřednictvím operací (metod).

Rozhraní objektu je množina operací, které nabízí objekt k použití pro jiné objekty (nebo externí agenty). Objekty jsou známy jiným objektům pouze prostřednictvím svého rozhraní. Objekt má i své vnitřní – interní operace, které slouží k udržení vnitřní konzistence (stavu) objektu.

Objekt může poskytovat **více rozhraní** – mít více rolí, podle kontextu ve kterém se nachází. Stejně jako v reálném světě člověk vystupuje v různých rolích podle toho, v jakém kontextu se právě nachází (v zaměstnání se nachází v roli pracovníka, doma manželem, v automobilu řidičem)

Objekty tak **odbourávají** nevýhodu strukturálních metod, spočívající ve vzájemné izolaci funkční a datové vrstvy.

Objekty spolupracují proto, aby společně mohly vykonávat funkce poskytované systémem, viz modely spolupráce.

Operace určující chování jsou definovány a rozpoznávány svojí signaturou – názvem, seznamem parametrů a návratových hodnot.
Objekt přijme zprávu a vykoná operaci, jejíž signatura je shodná se signaturou zprávy.

Diagram případů užití (Use Case Diagram)

Diagram případů užití vymezuje hranice systému specifikací **„funkční obálky“**, prostřednictvím které systém komunikuje se svým okolím – viz. kontextový diagram. Definování hranice systému.

Z vnějšího pohledu představuje soubor všech případů užití **úplný popis funkčnosti systému,** (vše co by měl systém umět, implementovat).

Případ užití (Use Case)
Element modelu případ užití specifikuje logicky ucelenou část funkcionality systému, tzv scénář dialogu uživatele se systémem.

Důležitou součástí případu užití je **popis funkčnosti**, nejčastěji ve formě připojeného textu rozděleného do jednotlivých kroků.

Popis – specifikace případu užití
Pro specifikaci užití neexistuje žádný standard UML.
Doporučení - strukturovaný text, pevně daná šablona:

- **Název** – slovesná vazba
- **ID** – strukturální kódové označení
- **Vstupní podmínky** – podmínky, kritéria, která musí být splněna ještě před „spuštěním“ případu užití, omezení stavu systému
- **Popis scénáře** – tok událostí, jednotlivé kroky případu. Jednoduchá sekvence číslovaných kroků, co dělá aktor, co dělá systém.
- **Následné podmínky** – podmínky, kritéria, která musí být splněna na konci případu.

Scénář - tok událostí je zachycen prostým textem, důraz na přesnost vyjadřování, možnost použít metajazyk (pseudokód). Textu rozumí uživatel i programátor.

Varianta přímá – jednoduchý, nevětvený, bezproblémový scénář s obsloužením mimořádných stavů a chyb - vyjímek v pod-scénářích, v alternativních tocích událostí. Rozdělení do vedlejších aktivit, pod-scénářů.

Varianta větvení - pro zápis větvení se používají „srozumitelná“ klíčová slova (*když-potom-jinak*), opakování (*Pro).*
Jsou názory, že uvnitř případů užití by nemělo docházet k větvení, sekvenční scénář.

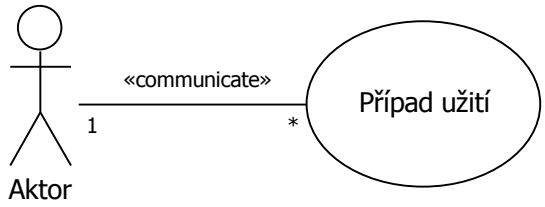
Alternativní toky, scénaře – varianty, které se mohou vyskytnou v předem nedefinovaných okamžicích, není pro ně definován bod volání, ale pouze podmínka, při jejímž splnění se přejde na alternativní tok, např. Uživatel „násilně“ ukončí tok událostí – ukončí Zapůjčení knihy (stisknutím tlačítka „storno“).

Aktor (Actor)
Element modelu aktor vyjadřuje prvek okolí systému, který se systémem komunikuje, přijímá nebo poskytuje systému informace.
Aktor je abstrakcí pro kohokoliv (uživatelé, skupiny uživatelů, …) nebo cokoliv (spolupracující systémy, subsystémy, čas…), co je považováno za **externí** z pohledu modelovaného systému.

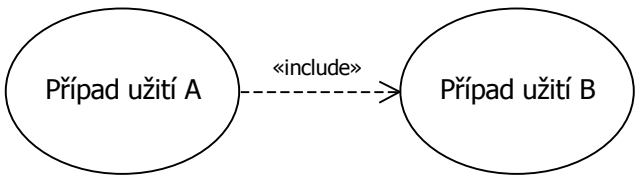
Účelem zavedení elementu modelu aktor je možnost **popisu okolí systému** a možnost **identifikace** všech případů užití z vnějšího pohledu.

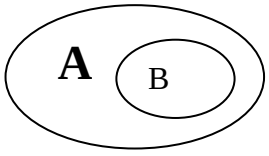
Identifikace případů užití:
Dekompozice systému z pohledu jednotlivých aktérů a pro každého z nich postupně rozpracovat případy užití, které vykonává. Funkční rozklad na elementy, které umožňující odhad pracnosti a tím i celkového času realizace systému.

Vazba asociace (Association)
Vazba *asociace* vyjadřuje nutnou interakci mezi aktorem a případem užití.



Vazba obsahuje, zahrnuje (Include Relationship)
Vazba vyjadřuje zahrnutí části funkcionality popsané jedním případem užití do jiného případu užití. **Z** vnějšího pohledu se však jedná pouze o jeden případ užití (orientace je vyjádřena šipkou k zahrnovanému případu užití, B je vytknuto z A, A zahrnuje B, B může být použito opakovaně – vytknuto z X).





V případu užití **A** (zahrnujícím) musí být určen bod volání, kam má být případ **B** zahrnut. Syntaxe „include“ s tak podobá volání funkce – řízení je přesunuto na B a po jeho ukončení (včetně návratové hodnoty) se vrací do **A**. Případ A je bez B neúplný.

Zavedení opětovné použitelnosti není v žádném případě novinkou a je obecným a dobře známým jevem v programování a obecněji v modelování SW. Můžeme namátkou jmenovat následující příklady: volání funkcí ve strukturálním programování, normalizace databáze, dědění a jiné interakce mezi třídami (např. asociace), obecně interakce mezi prvky modelu v UML, kdy jeden prvek používá druhý prvek, re - use.

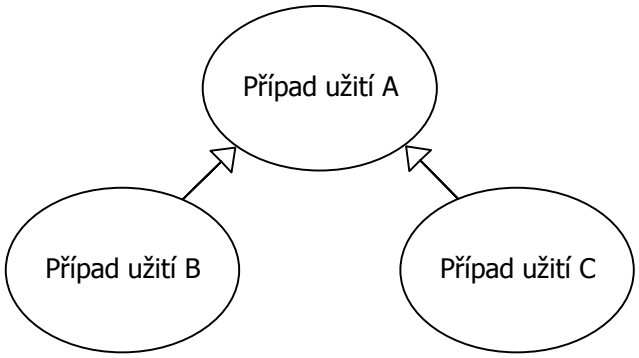
Vazba rozšiřuje (Extend Relationship)
Vazba *rozšiřuje* vyjadřuje možnost **alternativního rozšíření funkcionality** popsané jedním případem užití funkcionalitou popsanou separátním případem užití.

Na rozdíl od konstrukce s vazbou obsahuje musí být rozšiřovaný případ užití **proveditelný samostatně** (orientace je vyjádřena šipkou k rozšiřovanému případu užití).

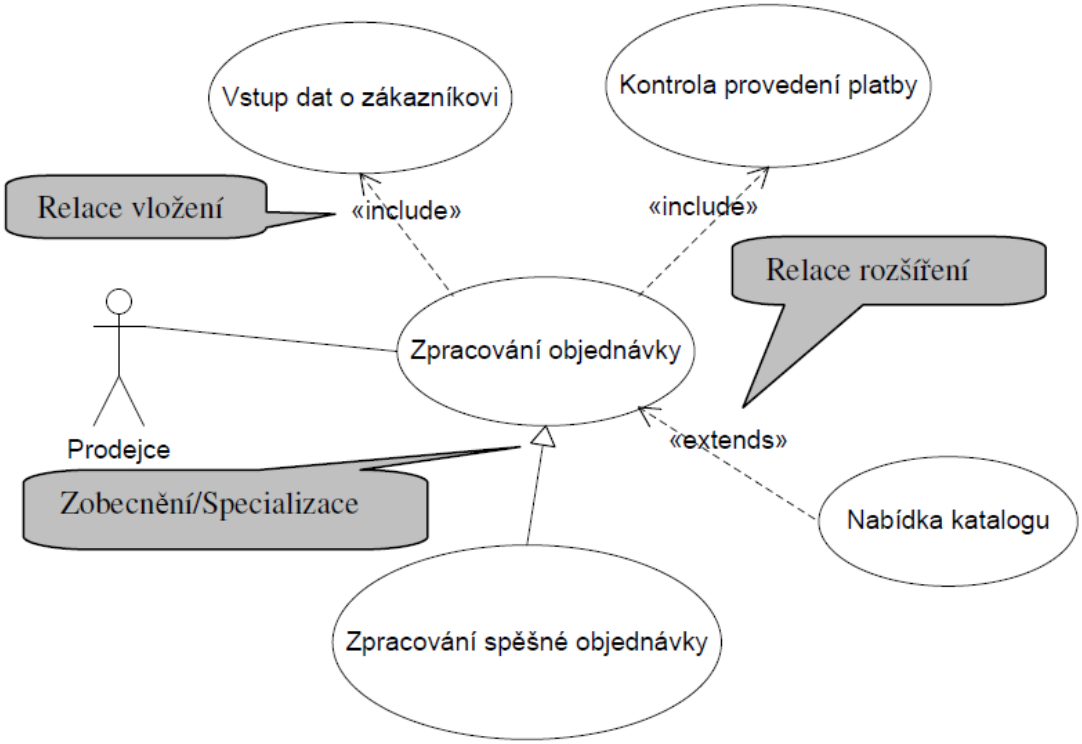
Případ A je proveditelný i bez případu B – rozdíl oproti relaci „include“. A má pouze připraven bod – **zásuvný modul** pro možné (např. v budoucnu specifikované) rozšíření, V předem definovaném bodě - bodech (extensit points) se vkládá rozšiřující funkcionalita – chování (insertion segment).

Podmíněné rozšíření - případ B rozšiřuje případ A pouze v případě splnění jisté podmínky: Např. případ Zapůjčit knihu je rozšířen o případ Rezervace Knihy, jestliže neexistuje pro danou knihu volný exemplář a Čtenář si chce na knihu udělat rezervaci.

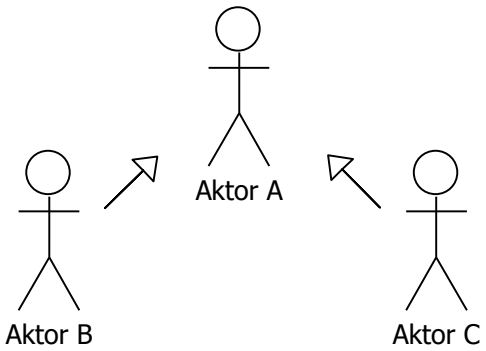
Vazba generaliace (Generalization)
Vazba *generalizace* je vyjádřením vztahu mezi obecným elementem (Parent) a specifickým elementem (Child), který přidává k jeho definici další informace, je tedy bližší specifikací (specializací) obecného elementu.



Zvyšuje nepřehlednost modelu, problém z popisem scénáře (popis dědičnost).



Relace mezi případy užití je třeba používat „střídmě“ a jenom tam, kde to zjednodušuje model (více-násobné užití, rozklad příliš rozsáhlých případů). Mnoho relací „include“ – mnoho nesamostatných případů užití.



Shrnutí:
Případ užití je scénář – sekvence dialogů uživatele se systémem
Případ užití je vždy iniciován aktérem
Případ užití vyjadřuje co, (ale nikoliv jak) budoucí systém nabídne uživateli.
Případy užití reprezentují vnější pohled na systém.

Řízení projektu na základě případů užití.
Jednotlivé případy užití představují elementární jednotky práce – úkoly zadávané členům týmu.
Na základě priorit případů užití lze plánovat pořadí vývoje systému.
Na základě odhadu časové náročnosti jednotlivých případů užití lze odhadnout, řídit a kontrolovat celý projekt.

Vztah případu užití a požadavků kladených na IS.

Případy užití nezachytí tzv. nefunkční požadavky uživatelů, jako např.:

- dodržení určitých standardů
- zabezpečení systému
- použitá technologie

Doporučuje se klást velký důraz na sběr, definici a řízení požadavků na IS (reguirements engineering). Vhodný výběr a zapojení všech skupin budoucích uživatelů.

Modely objektové spolupráce

Pomocí analytických tříd se modeluje statická struktura systému. Podstata specifikace dynamického chování objektově orientovaného systému spočívá v modelování komunikace - interakce mezi objekty.

Interakce mezi objekty je realizována tím, že si objekty posílají zprávy, na které odpovídají pomocí předem připravených operací (předem připravených metod).

Otázka zní, jakými operacemi vybavit objekty, tak aby dostály svým odpovědnostem a vzájemnou interakcí objektů byla zajištěna, pokryta požadovaná funkcionalita celého systému.

Objekty (třídy) mají mít málo jasně definovaných operací, které zajišťují jejich odpovědnost. Mají mít operace, které od nich očekáváme.

Odpověď – realizací případů užití. V případech užití je popsána funkcionalita systému.

Případy užití realizujeme tak, že pro ně hledáme třídy - objekty a ty vybavujeme metodami tak, abychom jejich vzájemnou interakcí realizovali scénář případů užití.

Pro modelování spolupráce objektů jsou v UML dva modely – diagramy:

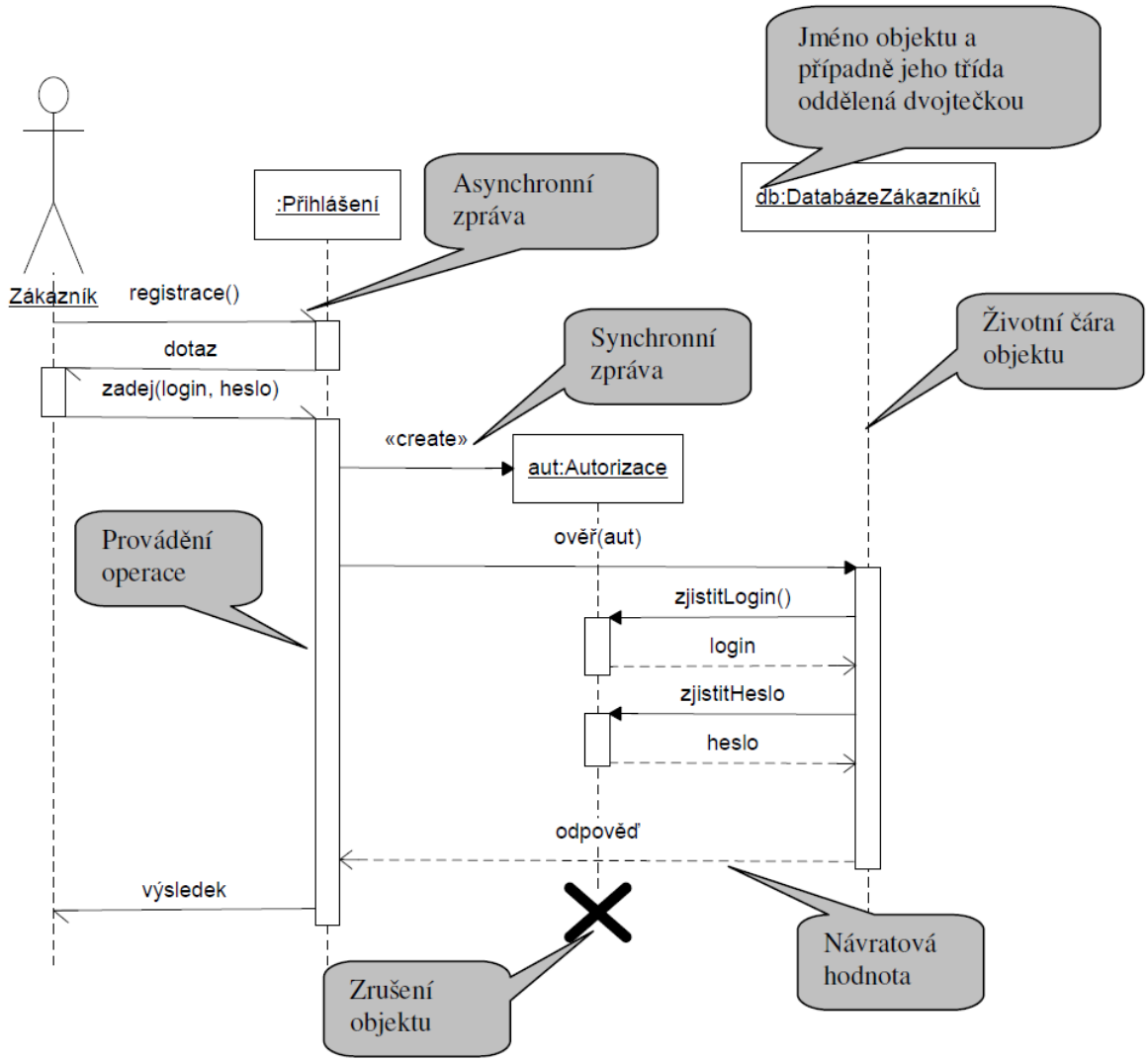
- Sekvenční diagram – sequence diagram
- Diagram objektové spolupráce – collaboration diagrams

Sekvenční diagramy
Dokumentují spolupráci objektů na scénáři případu užití. Kladou přitom důraz na časový aspekt komunikace. Dokumentují objekty a zprávy, které si objekty posílají při řešení scénáře.

Jsou vhodné pro popis scénáře při komunikaci s uživateli.

Diagram je tvořen objekty uspořádanými do sloupců a šipky mezi nimi odpovídají vzájemně si zasílaným zprávám.

Zprávy mohou být **synchronní** nebo **asynchronní**. V případe synchronních zpráv odesílatel čeká na odpověď (odezvu) adresáta, v případě asynchronní zprávy odesílatel nečeká na odpověď a pokračuje ve vykonávání své činnosti. Souvislé provádění nějaké činnosti (operace) se v sekvenčním diagram vyjadřuje svisle orientovaným obdélníkem. Odezvu adresáta lze modelovat tzv. návratovou zprávou (přerušovaná čára). Tok času probíhá ve směru shora dolu.



Dobrý návrh má rovnoměrně distribuovanou inteligenci ve formě „jednoduchých“ operací, metod v rozdílných třídách.

Sekvenční diagramy používáme pro „stěžejní“ případy užití.
Složitější modely – násobné objekty, větvení a iterace komplikují přehlednost modelů.

Datové modelování

Jedna ze základních funkcí IS – ukládání a následné zpracování informací.

Ve fázi návrhu IS je třeba rozhodnout, jak budou objekty, 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.

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 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 – složitější odpověď – žijeme v epoše spalovacích motorů a relačních databází.

Musíme perzistentní objekty mapovat – transformovat na relace – tabulky relační databáze.
Je objektový koncept principiálně nevhodný pro perzistenci?

Datové modelování
Vojtěch Merunka
Alfa Publishing,

Cílem datového modelování je navrhnout „kvalitní“ datovou strukturu a databázový systém pro konkrétní IS.

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

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

Dva pohledy na data – logický, konceptuální a fyzický model.

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

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.

Entita - datový objekt, čistě datový pohled z hlediska perzistence, na rozdíl od objektového konceptu třídy se vyloučí, abstrahuje chování objektu. Určující je schopnost databázového stroje uložit (implementovat) vlastnosti objektů.

Entita bude ve fyzickém modelu **tabulkou** relační databáze. Tabulka je relací – obsahuje n-tice (řádky), jestliže žádné dva řádky tabulky nejsou shodné, viz primární klíč.

Entita – předchůdce objektového konceptu třídy.

ERA - model (Chen, 1976)

- E-entita, objekty které nás zajímají
- R-relace, vztahy mezi objekty
- A-atribut, vlastnosti - atributy objektů resp. vztahů, které potřebujeme k dosažení daného cíle ?

Terminologie:

Typ entity - výskyt entity
Třída - instance třídy (objekt)
Typ proměnné - proměnná

Při analýze pracujeme s typy entit

Vztahy mezi entitami:

Viz vztahy mezi třídami v objektovém modelu.
Vztahy mezi entitami, které lze implementovat v relační databázi

- Kardinalita, parcialita – maximální a minimální počet výskytů ve vztahu
- odvoditelnost

Odvoditelnost
Rozdělení na primární a sekundární vazby (odvoditelné). Nelze zachytit všechny vztahy mezi objekty, je třeba nalézt primární vztahy, které nelze odvodit.

Parcialita
Při výskytu jedné entity ve vztahu musí existovat výskyt druhé entity. Členství ve vztahu - entity zapojené do vztahu jsou členy vztahu - povinné a nepovinné členství.

Atributy

Entity jsou vymezeny (popsány) množinou atributů. Atributy popisují vlastnosti entit, které potřebujeme k dosažení daného cíle a které potřebujeme zachovat - perzistence.

Sémantika ERA modelu je dána integritními omezeními – vlastnostmi entit, jejich vazeb a jejich atributů.

Primární klíč
Klíč je **minimální** množina atributů, jejichž hodnoty jednoznačně určují výskyt entity. Klíčů může být více, vybírá se jeden, tzv. primární klíč.
Skládá-li se primární klíč z více atributů – tzv. složený klíč, nemůžeme žádný z nich vyloučit, aniž bychom porušili podmínku jednoznačné identifikace entity.

Silné a slabé entity
V souvislosti s jednoznačnou identifikací entity se rozlišují silné a slabé - závislé entity.

Existenční závislost
Viz kompozitní vazba asociace

Identifikační závislost
Slabé entity nemají svoji vlastní identitu, „svůj“ primární klíč. Primární klíč slabých entit je složený klíč z primárních klíčů řídících entit, viz vazební entita.

Alternativní klíč
Jeden nebo více atributů, které také jednoznačně určují výskyt entity. Jsou možnou alternativní volbou pro primární klíč.

Problematika volby primárního klíče.
Zásadní otázka datového modelu, má dalekosáhlé implementační důsledky.

Dvě varianty:

- Volba přirozeného, inteligentního, významového (business-related) atributu, např. pro čtenáře -čísla čtenáře přidělené knihovnou, pro knihu její ISBN.
- Systémem generovaný, umělý klíč (surrogate) – obvykle jednoznačné číslo přidělené fyzickému záznamu v okamžiku jeho založení do databáze.

Systémový klíč na rozdíl od přirozeného nemá žádný explicitní význam – je to jeho výhoda, ale i nevýhoda.

Význam primárních klíčů ve fyzickém datovém modelu - implicitní vyjádření vazby, relace mezi tabulkami (větami v tabulce).

Cizí (vazební) klíč
Jedná se o atribut nebo skupinu atributů, jejichž hodnota je buď prázdná, nebo musí být obsažena jako hodnota primárního klíče jiné relace. Pomocí cizích klíčů se realizuje implicitní vazba mezi tabulkami fyzického modelu, obvykle vazba kardinality 1: N. Říkáme, že tabulky jsou ve vztahu Master-Detail, Parent-chield.

Vazební klíče se (automaticky) generují při přechodu z konceptuálního na fyzický datový model. Implementují vazby a jejich vlastnosti mezi entitami na fyzické úrovni.

Primární klíče parent-tabulek vystupují jako cizí klíče do chield-tabulek. Zajišťují tzv. vazební integritu. Rozhodující pro volbu mezi přirozeným a systémovým klíčem.

Viz Case

Výhody systémových klíčů:

- Automaticky generované, stabilní, po celou dobu existence relace, záznamu. Vstupují do vazeb jako cizí klíče, vytvářejí stabilní referenční integritu.
- Celočíselný formát systémových klíču – nejvhodnější formát pro vytváření indexů, rychlý výběr záznamů, databáze obvykle automaticky generují indexy pro primární i cizí klíče.

Nevýhody systémových klíčů:

- V závislých chield-tabulkách není žádná informace, např. kód, číslo čtenáře z parent-tabulky. Informace z parent-tabulky musíme „dotáhnou“, viz konstrukce select … join.

Nevýhody přirozených klíčů

- Zásadní problémy při vynucené změně klíče, např. přečíslování čtenářů v knihovně, sloučení dvou knihoven.
- Závislost tabulek vede ke složeným klíčům, problémy s indexováním.

Dobrá praxe, vývojový vzor (Hibernate) – používat systémové, vnitřní klíče.
Špatná praxe – složený klíč z atributů různých typů (textové atributy proměnné délky).
Nevhodná indexace – neúčinné, velké indexové soubory.

Závislost atributů, proces normalizace.

Analýza významové (sémantické) závislost mezi atributy (sloupci tabulek ve fyzickém modelu) či jejich množinami a její případné odstranění.

Řekneme, že atribut B je funkčně závislý na atributu A, pokud hodnota atributu A určuje hodnotu atributu B (z hodnoty A zjistíme jednu a právě jednu hodnotu B).

Odvozené, vypočtené atributy:

- Výpůjčka – datum zapůjčení, datum vrácení, → délka výpůjčky

Strukturální závislost:

- Třída → Ročník (existují další závislosti v tabulce)

Login	Jméno	Specializace	Třída	Ročník
NOVAKA	Novák	Podnikové informační systémy	I3	3
NOVAKJ	Novák	Služby knihoven	K2	2
PECHOVA	Pechová	Služby knihoven	K2	2
RASKOVA	Rašková	Podnikové informační systémy	B1	1
SOUMAR	Soumar	Služby muzeí a galerií	M2	2

Normalizace datového modelu

Normalizace fyzického datového modelu:

- úprava tabulek s cílem vyloučit redundaci (nadbytečnost) dat a zajistit konzistenci základních transakcí – změna hodnot atributů, přidání a smazání záznamů (změna třídy žáka v předchozí tabulce).

Nadbytečnost – každý atribut by se měl v modelu objevit pouze 1x–oprávněnost požadavku ?, např. adresa zaměstnance, dodavatele, odběratele… .

Celková optimalizace datového modelu. Relační databáze se dají navrhnout tak špatně, že se ztratí celý výkon databázového stroje. Obecně se při normalizaci tabulky s více sloupci rozpadají na více „provázaných“ tabulek s méně sloupci.
Proces rozpadu je třeba zastavit „na rozumné“ hladině a „strpět“ jistou nadbytečnost dat.

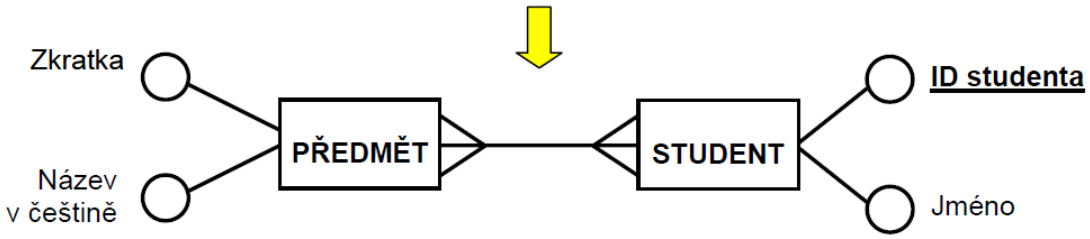
1. normální forma (1NF)

- každý atribut (sloupec) tabulky musí obsahovat pouze jedenu, dále nedělitelnou atomickou hodnotu, bez vnitřní struktury.
- tabulka nesmí obsahovat sloupec s vícenásobnou hodnotou, např. znalost cizího jazyka -Angličtina, Němčina, Francouzština.

2. normální forma (2NF)
Řeší problém vnitřní funkční závislosti. Tabulky obsahují pouze takové sloupce, které jsou funkčně (významově) závislé na celém primárním klíči. Neobsahuje sloupec, který závisí např. pouze na části složeného klíče nebo je na primárním klíči dokonce nezávislý.

Př.: Evidence předmětů zapsaných jednotlivými studenty. Název ale ani zkratka předmětu nejsou funkčně závislé na ID studenta. Řešením je rozklad tabulky do dvou tabulek – Studen a předmět. Evidence zapsaných předmětů je implementována vazební tabulkou. Nevýhody původního řešení – vztah – relace mezi studentem a zapsaným předmětem je „fixován“ v tabulce. Jak řešit např. změnu zkratky? Vyskytuje se (je utopena) u všech studentů

ID studenta	Jméno	Zkratka	Název v češtině
123	Petr Blažek	ANG	Angličtina
124	Jan Nový	MAT	Matematika
125	Eva Kotíková	MAT	Matematika



3. normální forma (3NF)
Řeší problém vnitřní, tranzitivní závislost sloupců. Žádný neklíčový sloupec tabulky nesmí být závislý na jiném neklíčovém sloupci. např. název předmětu je pevně fixována na kód.

Realizace vazby typu M:N

Realizace vazby znamená podchytit skutečnost, že daná entita z jedné entitní množiny je spojena s danou entitou druhé množiny, bez ohledu na to, že je spojena i s jinými entitami. Toho lze vždy dosáhnout vazební entitou, která obsahuje dva vazební atributy, jeden obsahuje primární klíč z prvé entity a druhý primární klíč z druhé entity. Primárním klíčem vazební entity je takto zkonstruovaná dvojice. Otázka rozkladu vazby M:N na více nezávislých vazeb 1:N.

Formát atributů – datových položek, sloupců tabulek.
Přípustné formáty atributů na konceptuální a fyzické úrovni, viz CASE. Na fyzické úrovni pro konkrétní databázový stroj volíme přípustné datové typy.

Nad atributy, datovými položkami můžeme volit další omezení, tzv. doménová integrita – výčet přípustných hodnot, min. a max. hodnotu.
Zásadní rozhodnutí – zda atribut může mít neurčenou, nezadanou hodnotu, tzv. NULL, viz syntaxe SQL.

Uložené procedury, triggery

Objektově relační mapování

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

Historie UML

(Jak to v normalizačním procesu chodí). V návaznosti na vznik prvních objektově orientovaných jazyků se začínají v polovině **70-tých let** objevovat i první objektově orientované metody analýzy a návrhu (v roce 1994 více než padesát).

Stávající strukturální metody přestávají vyhovovat. Malá soudržnost datového a funkčního modelu, **růst složitosti systémů**, manipulace se stejnými daty se realizovala na mnoha místech programového kódu.

V **90-tých letech** se začínají objevovat metody, které mezi ostatními zaujímají dominantní postavení, každá má své přednosti a nedostatky:

- metoda Boochova (Booch’93)
- Jacobsonova (OOSE – Object Oriented Software Engineering)
  Přístup založený na tvorbě případů užití (Use Case), který se ukázal jako velmi vhodný v úvodních fázích analýzy, především v analýze požadavků na navrhovaný softwarový systém.
- Rumbaughova (OMT-2 – Object Modeling Technique).
  Silná analytická část.

Z těchto tři metod postupem času vznikl jednotný modelovací jazyku (UML – Unified Modeling Language).

Vývoj UML byl zahájen v druhé polovině **roku 1994**, když Grady Booch a Jim Rumbaugh pod křídly společnosti Rational Software Corporation zahájili práce na unifikaci vedoucí ke sjednocení svých metod.

V roce 1995 se k Rational Software Corporaton připojil i Ivar Jacobson a jeho společnost Objectory Company a byla zahájena integrace metody OOSE.

Počátkem **roku 1996** se objevuje jednotný modelovací jazyk verze 0.9 (UML – Unified Modeling Language).

Během roku 1996 se rovněž začalo konstituovat konsorcium **UML Partners Consorcium**, jehož členy se kromě Rational Software Corporation stala řada významných světových firem (Digital Equipment Corp., Hewlett-Packard Company, IBM Corporation, Microsoft Corporation, Oracle Corporation, …) – nestát stranou, prosazení vlastních zájmů.

V lednu **roku 1997** byla uvedena **UML verze 1.0**, avšak práce pokračovaly dále.

Další cesta vývoje vedla především ke zpřesnění definice UML a k zapracování řady podnětů na další rozšíření UML (problematika business modelování, jednotný jazyk pro zápis podmínek a omezení, …).

UML získala postupně **dominantní postavení** v oblasti metod objektově orientované analýzy a návrhu a toto postavení bylo formalizováno ve druhé polovině roku 1997 mezi schválené a podporované technologie **organizace OMG** (Object Management Group, Inc.), která sdružuje více než osm set významných organizací a společností z počítačového světa.

Základní charakteristika UML

Unifikovaný modelovací **jazyk** UML je podle specifikace charakterizován jako jazyk pro vymezení, vytvoření, znázornění a dokumentování softwarových systémů, jakož i pro business modelování a další nesoftwarové systémy.

Byť lze proti UML vznést řadu připomínek, je otázkou, zda reálně existovala jiná alternativa tohoto integračního procesu. Bylo by v silách jednotlivce nebo úzké skupiny jednotlivců vytvořit novou metodiku tak, aby si získala celosvětovou akceptaci a dominantní postavení jako UML.

Nedostatek – absence procesního modelování, např. diagram hierarchie procesů. Modelování procesů dle nového standardu OMG BPMN (Object Management Group – Business Process Modeling Notation)

Specifikace UML

Úplná dokumentace jazyka UML čítá stovky stran (syntaxe a sémantika jazyka).

Základ specifikace UML je dán metamodelem UML, který je vlastně modelem samotného jazyka UML, přesněji modelem syntaktických pravidel.

V definici UML je zavedena čtyřvrstvá hierarchie modelů:

Vrstva	Popis	Příklad
Meta-metamodel	Definuje jazyk pro specifikaci metamodelu.	MetaClass MetaAttribute
Metamodel	Je instancí meta-metamodelu. Definuje jazyk pro specifikaci modelu.	Class (instance od MetaClass) Attribute (instance od MetaAttribute)
Model	Je instancí metamodelu. Definuje jazyk pro popis informační domény.	Město (instance od Class) Počet obyvatel (instance od Attribute)
Systém (realizace modelu)	Je instancí modelu. Definuje specifickou informační doménu.	Plzeň (instance od Město) 180 000 (hodnota atributu Počet obyvatel)

Vzhledem k tomu, že UML byl navržen jako jazyk pro vizuální modelování, je zaveden pojem **diagramu a vizuálních elementů**, které jsou grafickou reprezentací elementů modelu v diagramu.

Na model systému lze nahlížet z **různých pohledů**, které odpovídají určitým specifickým aspektům pohledu na systém.

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

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

Analytické modely se zabývají zejména otázkou "**CO**" by měl systém dělat.
Návrhové modely popisují dekompozici systému na **programátorsky zvládnutelné části** - zabývají se otázkou "**JAK**" by to mělo být uděláno.
Implementační modely dokumentují implementaci.

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

Dělení na dvě základní skupiny:
Diagramy struktury (Structural Diagrams) zachycující statické strukturální aspekty modelovaného systému
Diagramy chování (Behavioral Diagrams) zachycující dynamické aspekty modelovaného systému.

Diagramy struktury:

- **Diagram tříd** (Class Diagram) zobrazuje třídy a jejich vztahy.
- **Objektový diagram** (Object Diagram) zobrazuje objekty (instance tříd) a jejich vztahy v určitém okamžiku v čase.
- **Komponentový diagram** (Component Diagram) zobrazuje strukturu realizace softwarového systému zachycenou pomocí softwarových komponent, rozhraní (Interface) a jejich vztahů.
- **Diagram rozmístění zdrojů** (Deployment Diagram) zobrazuje strukturu realizace systému zachycenou pomocí softwarových prvků a (softwarových komponent existujících v run-time systému), hardwarových prvků – uzlů (Node) a jejich vztahů.

Diagramy chování:

- **Diagram případů užití** (Use Case Diagram), popisuje systém jako seznam případů užití (Use Case).
- **Sekvenční diagram** (Sequence Diagram), popisuje spolupráci objektů prostřednictvím sekvence zasílaných zpráv.
- **Diagram spolupráce** (Collaboration Diagram), vyjadřuje stejnou informaci jako sekvenční diagram, pouze v odlišném grafickém zobrazení.
- **Stavový diagram** (Statechart Diagram), popisuje stavy objektů a přechody mezi stavy.
- **Diagram aktivit** (Activity Diagram), popisuje posloupnosti aktivit.

Obecné rozšiřující mechanismy (General Extension Mechanisms)

V UML není možno zásadním způsobem modifikovat stávající nebo vytvářet nové elementy modelu. Existuje však několik rozšiřujících mechanismů společných pro všechny elementy modelu UML, které umožňují jejich uživatelské přizpůsobení.

Připojená hodnota (Tagged Value)
Mechanismus připojených hodnot umožňuje připojit k libovolnému elementu modelu vlastní pojmenované hodnoty (dvojice: název a hodnota)

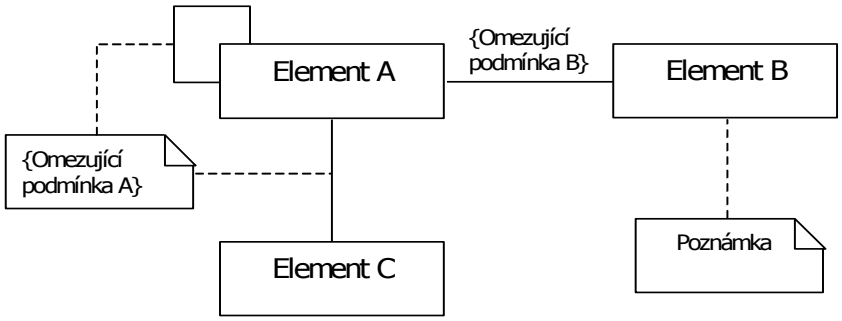
Stereotyp (Stereotype)

Mechanismus stereotypů umožňuje zavést **vlastní kategorizaci** standardních elementů modelu přiřazením stereotypu elementu modelu, a to včetně změny standardní grafické podoby odpovídajícího vizuálního elementu (např. různé typy – kategorie asociací).

Poznámka (Note)
Ke každému elementu modelu lze přiřadit poznámku ve formě volně psaného textu. Poznámka je zobrazena textovým polem připojeným přerušovanou čarou k příslušnému vizuálnímu elementu.

Omezení (Constraint)
Mechanismus omezení umožňuje přiřadit k jednomu nebo více elementům modelu uživatelsky definovanou omezující podmínku, kterou není možno vyjádřit pomocí standardních elementů modelu UML.

Správnost modelu je podmíněna splněním těchto podmínek. UML doporučuje využívat pro jejich zápis OCL (Object Constraint Language), který je součástí definice UML.



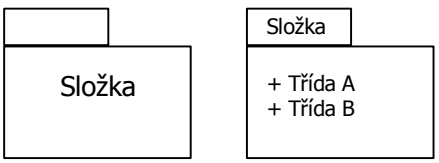
Organizace modelu

Složky (Package) jsou nástrojem UML umožňujícím organizaci modelu, **seskupováním elementů** modelu do složek.

Složky jsou sami o sobě elementy modelu, a tedy je možno pomocí tohoto mechanismu vytvářet víceúrovňové **hierarchické struktury** složek.

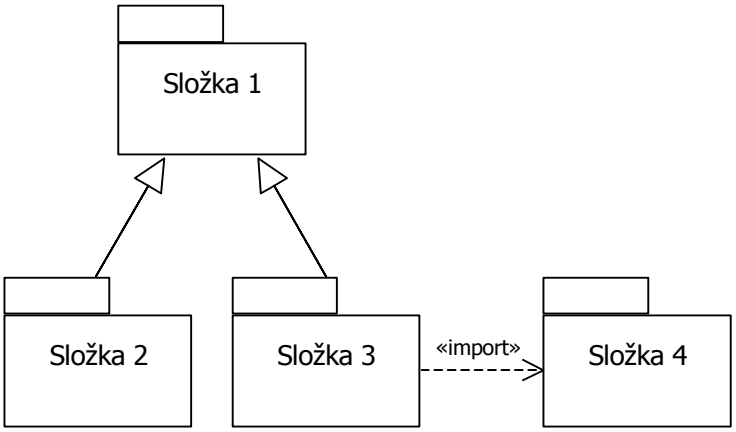
Mezi složkou a elementem modelu je **vztah vlastnictví**, což znamená, že každý element modelu se v daném okamžiku může nacházet právě v jedné složce.

Složky definují **názvové prostory** (Name Space), v rámci kterých musí být dodržena



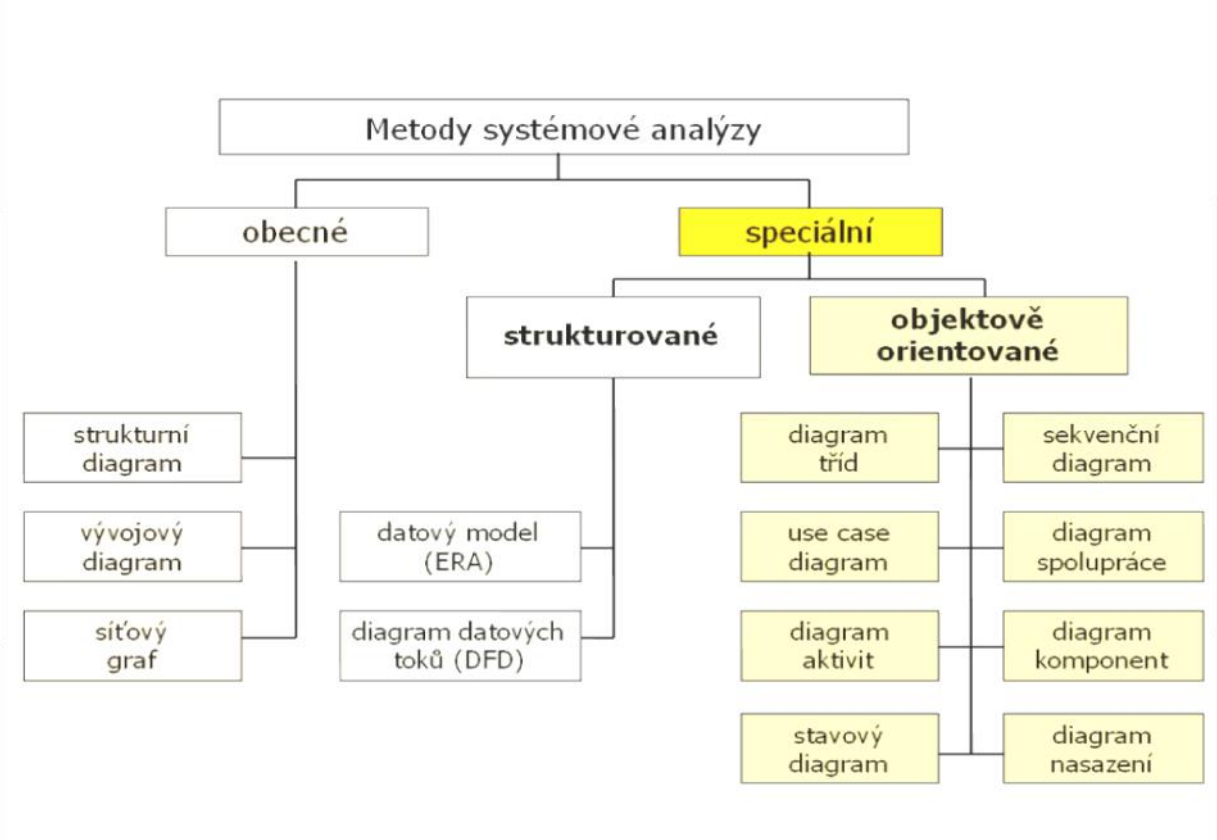
jednoznačnost pojmenování elementů modelu.
Mezi složkami je možno použít dva základní typy vazeb:

- vazbu závislosti se stereotypem
- vazbu generalizace/specializace pro vyjádření vlastnosti dědičnosti



Specifikace výměnných formátů

Součástí definice UML je i specifikace výměnných formátů, sloužících např. pro přenos zápisů v UML mezi různými nástroji. Jako příklad lze uvést formát CORBA IDL (Interface Definition Language), či XMI (XML Metadata Interchange).



Procesní modelování

Procesní modelování není standardní součástí UML. Metodiky využívají procesní modelování jako úvodní krok analytických prací a složí k modelování firemních procesů.
Diagram procesů modeluje *business* procesy v podniku. Jednotlivé procesy mohou být dále děleny do pod-procesů, jednotlivé nezávislé procesy jsou vzájemně synchronizovány prostřednictvím spojení výstupních stavů a počátečních událostí (výstupní stav jednoho procesu může být vstupní událostí procesu jiného).

Diagram hierarchie procesů

První typ procesního modelování je Diagram hierarchie procesů, který řeší procesní rozpad – dekompozici systému.

Vlastnosti a možnosti CASE

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.

CASE čelí složitosti systému, která by bez jejich pomoci byla těžko zvládnutelná.

CASE zajišťuje kvalitu procesů vývoje softwaru (díky použitým metodikám a odhalování chyb při jejich použití)**.**

CASE zajišťuje značnou úsporu času (a tedy nákladů) potřebného k vývoji systému**.**

CASE 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 specifikace 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.
- Upper CASE.
- Middle CASE.
- Lower CASE.
- Post CASE.

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

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.

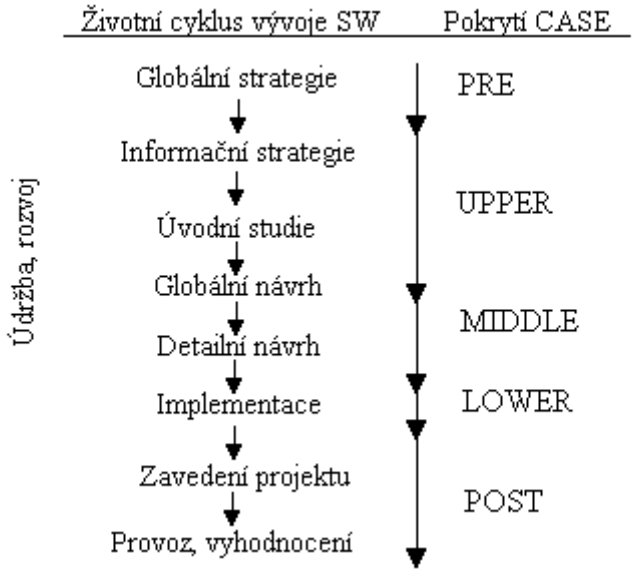
Middle CASE
Tento druh CASE nástrojů je základem všech komerčně dodávaných nástrojů.

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.

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.
