

1. Úvod

Nasledujúca publikácia sa zaobrá najstarším nástrojom a postupom rozvíjajúcim ľudské predstavy a skúmajúcim reálne a imaginárne systémy. Týmto nástrojom je model a postupom modelovanie (vytváranie modelu) a simulácia (experimentovanie s modelom). Za všeobecne známy a pravdepodobne najstarší nástroj môžeme model považovať z jednoduchého dôvodu: modelom je v podstate takmer každá hračka, simulácia a hranie majú k sebe tiež veľmi blízko. Najstaršie, dôverne známe nástroje a postupy sú v ďalšom teste popisované vo vzťahu k najmladším a najdynamickejšie sa rozvíjajúcim systémom – programovým (softvérovým) systémom.

Publikácia integruje cez pohľad modelovania dve úzko súvisiace činnosti: riadenie a realizáciu softvérových projektov. Pretože realizácia projektov (analýza, návrh a implementácia systémov) je predmetom riadenia projektov, existuje aj vzťah medzi modelmi používanými v oboch týchto činnostach. Najčastejším obsahom softvérových projektov v súčasnosti sú informačné systémy – preto predmetom záujmu sú modely týkajúce sa tejto špeciálnej skupiny programov. Pod pojmom systém v ďalšom teste, pokiaľ nie je uvedená bližšia špecifikácia, uvažujeme informačný systém.

Cieľom práce nie je urobiť prehľad všetkých v súčasnosti používaných metodológií, metód a nástrojov analýzy, návrhu a implementácie – nie je to možné, vzhľadom na rozsah, ale ani nutné, vzhľadom na to, že existuje dosť monografií v tejto oblasti. Taktiež práca neobsahuje žiadnen popis, ani prehľad existujúcich softvérových nástrojov a systémov pre podporu projektovania softvérových systémov. Hlavný zámer je prezentovať modelovanie a súvislosti medzi modelmi ako hlavný a účinný nástroj životného cyklu programových systémov, aj keď vzhľadom na rozsah, nie je možné postihnúť všetky jeho fázy. Vzhľadom na analogické problémy vývoja technických a programových prostriedkov a ich vzájomné úzke prepojenie, je možné problematiku modelovania vziať do úvahy. Vzhľadom na obe zložky počítačových systémov – technickú a programovú (zaujímavá integrovaná metodológia v tomto zmysle je uvedená v [16]). Práca je určená študentom predmetov softvérového inžinierstva a odborníkom z oblasti riadenia a realizácie softvérových projektov. Nezaobrá sa programovou implementáciou systémov, ale vzhľadom na používané pojmy predpokladá čitateľa základné znalosti zo štruktúrovaného, modulárneho a objektového programovania. Podobne sa predpokladajú základné znalosti zo základov informatiky, teórie množín, formálnych jazykov, automatov a grafov.

2. Modelovanie a softvérové systémy

Pre zvládnutie zložitých situácií, postupov a systémov sa úspešne využíva metóda zjednodušenia pohľadu na skúmaný predmet – vytváranie modelov. Zjednodušenie je možné dosiahnuť:

- uvažovaním len tých aspektov, ktoré sú z daného pohľadu podstatné – abstrakciou
- vymedzením hranic tej časti skúmaného predmetu, ktorej sa tieto aspekty týkajú - štruktúrovaním.

Obidva postupy veľmi úzko súvisia. Štruktúrovanie – atomizácia na presne definované elementy – je postup, ktorý v chouse fungujúcich a nefungujúcich, existujúcich

a imaginárnych systémov hľadá určitú usporiadanosť, pravidelnosť, opakovanie, ohriadenie. V skutočnosti tieto štruktúry nemusia vôbec existovať, alebo existujú v podstatne zložitejších vzájomných vzťahoch. Štruktúrovanie pomáha pochopiť zložité systémy a/alebo uľahčiť ich syntézu tým, že orientuje pozornosť iba na sledované vlastnosti. Asi nie je potrebné zdôrazniť, že výber týchto vlastností ovplyvní rozpoznávanie štruktúr a to v konečnom dôsledku môže výrazne ovplyvniť napr. vlastnosti syntetizovaného systému.

Proces štruktúrovania v sebe zahŕňa aj abstrakciu: aby v systéme bolo možné vymedziť určité štruktúry, je potrebné sledovať len určité vlastnosti. Abstrakcia samotná nemusí viesť k štruktúrovaniu – celý systém bude z určitého pohľadu uvažovaný ako jeden celok.

2.1. Štruktúrovanie softvérových systémov

Programový systém je možné štruktúrovať zhora-nadol na elementy podľa Obr. 2.1 (proti orientácii vyznačených šípek). Celý programový systém môže tvoriť práve jeden alebo viac programov. Program môže pozostávať z práve jednej programovej jednotky (hlavnej funkcie, hlavného programu) alebo viacerých programových jednotiek – podprogramov (procedúr, funkcií, makier). Zvolená úroveň štruktúrovania pri analýze a syntéze na **podsytémy, programy, moduly, triedy a podprogramy** závisí od zložitosti analyzovaného (navrhovaného, programovaného) systému a od zvolenej metódy analýzy, návrhu alebo implementácie-programovania. Pritom je možné vykonať štruktúrovanú analýzu (použitím nejakej metódy štruktúrovanej analýzy) a implementáciu vykonať objektovo-orientovaným spôsobom (OOP - Object-Oriented Programming) vytváraním tried, objektov a definovaním ich správania.

Programové systémy svojimi vlastnosťami (poskytovanými funkciami, službami prostredníctvom rôznych technických prostriedkov) dopĺňajú a/alebo nahradzajú iné systémy (napr. pre výpočty, modelovanie, plánovanie, poskytovanie informácií, riadenie a pod.). Reálne systémy (určené napr. na poskytovanie informácií a na riadenie iných systémov) sú obyčajne zložité, rozsiahle a veľmi komplexné.

Zložitosť a rozsiahlosť systémov prináša negatívne dôsledky v náraste nákladov na ich návrh, implementáciu, údržbu a inováciu, t.j. rastú náklady na celý ich životný cyklus. Väčšia zložitosť a rozsah sú spravidla príčinou aj väčšieho množstva rôznych skrytých chýb. Projektová dokumentácia rozsiahlych a zložitých systémov býva tiež rozsiahla. Pre údržbu a modifikáciu systému alebo jeho podsystémov je dôležitá dobrá dokumentácia k existujúcej verzii systému. Jej udržiavanie v konzistentnom stave so systémom nie je jednoduchá úloha.

Zaužívanými postupmi softvérového inžinierstva pre efektívne zvládnutie zložitosti analyzovaných, navrhovaných (a samozrejme tiež implementovaných) systémov sú dekompozícia, syntéza, modelovanie, simulácia a prototypovanie:

- **Dekompozícia** - rozklad systému na komponenty (štruktúry) vyhovujúce vopred špecifikovanej abstrakcii s cieľom vytvoriť zvládnuteľné pohľady na systém bez zbytočnej detailizácie na danej úrovni (napr. rozklad informačného systému na služby podľa toho, ako sa používajú v rámci organizačnej štruktúry organizácie).

- Syntéza** - zostavovanie systému z existujúcich komponentov známych vlastností, správania, štruktúry a podobne (napr. zostavenie modulov z volaní knižničných podprogramov).
- Modelovanie** - vytváranie abstraktívnych formálnych a semiformálnych modelov systému (napr. vytvorenie entitno-relačného dátového modelu systému, objektového modelu systému a pod.).
- Simulácia** - experimentovanie s abstraktívymi modelmi systému (napr. simulácia časovej odozvy viacužívateľského informačného systému na rôzne druhy dotazov a rôznu štruktúru bázy dát).
- Prototypovanie** - vytváranie funkčných prototypov systému a experimentovanie s nimi (napr. prototyp používateľského rozhrania).

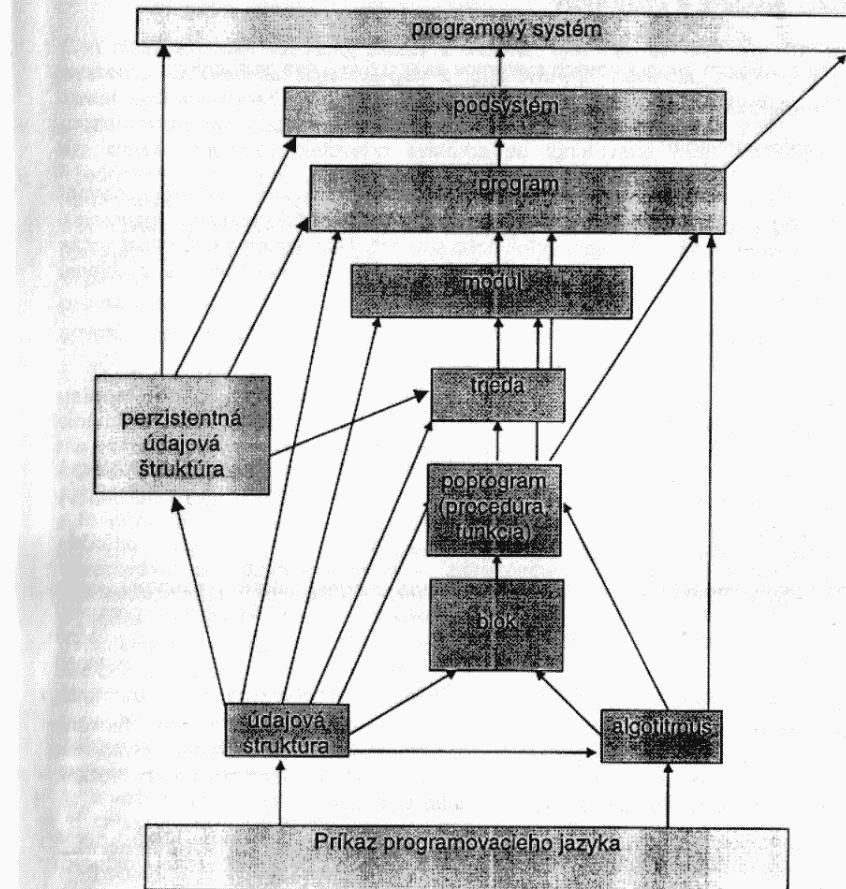
Dekompozícia, syntéza, modelovanie, simulácia a prototypovanie patria medzi základné inžinierske postupy aplikované pri projektovaní rozsiahlych programových systémov - sú to základné postupy softvérového inžinierstva.

Tieto postupy:

- používajú rôzne formálne a semiformálne nástroje modelovania
- sú obsahom rôznych **metód** použitia nástrojov pre analýzu, návrh alebo implementáciu systémov
- sú súčasťou rôznych **metodológií** podporujúcich viac fáz životného cyklu systémov.

Pri klasifikácii nástrojov na semiformálne a formálne sa uvádzajú často ako príklady semiformálnych nástrojov grafické prostriedky, napr diagram dátových tokov (DFD – Data Flow Diagram) a ako príklad formálneho nástroja niekterý z formálnych textových špecifikačných jazykov (Napr. jazyk Z, VDM a pod.). Ak formálnosť chápeme ako vlastnosť, ktorá umožňuje presný, syntakticky a sémanticky jednoznačný popis, potom aj DFD môžeme považovať za formálny prostriedok. Má jednoznačne definovanú syntaxu (syntax grafového jazyka) a nieje problém, v prípade potreby, dodefinovať jednoznačnú sémantiku pre všetky typy uzlov, hrán a vzájomných spojení. Táto sémantika môže byť vyjadrená pomocou nejakého programovacieho jazyka, jeho podmnožiny alebo pomocou formálneho špecifikačného jazyka. Nespornou výhodou grafických nástrojov oproti textovým formálnym špecifikačným jazykom je:

- vyššia zrozumiteľnosť, prehľadnosť grafickej špecifikácie oproti textovej pre všetky skupiny osôb zúčastnené na životnom cykle systému
- schopnosť grafických modelov prejsť od grafickej reprezentácie k podrobnej formálnej textovej špecifikácii detailizáciou zhora-nadol len v tých častiach systému, kde to je nutné z hľadiska preferovaných vlastností budúceho systému
- nezaťažovať zbytočnou presnosťou, pracnosťou a často aj rozsahom popisov modelovanie tých aspektov systému, ktoré postačuje verifikovať experimentovaním, simuláciou alebo testovaním prototypu.



Obr. 2.1 Štrukturalizácia programového systému

2.2. Modely a prototypy

V súvislosti s vývojom programových systémov sa používajú dva prístupy:

- modelovanie
- prototypovanie

Pri modelovaní sa využívajú prevažne **abstraktné modely** systémov, pri prototypovaní zase **prototypy** (v súčasnosti najčastejšie prototypy používateľského rozhrania). Prototypy a modely môžu navzájom veľmi úzko súvisieť. Táto vzájomná súvislosť môže byť v jednotlivých fázach životného cyklu výhodne využitá pre skvalitenie a zrýchlenie potrebných činností.

Používané modely systémov majú rozličnú úroveň abstrakcie. Podľa tejto úrovne abstrakcie ich môžme rozdeliť do nasledovných troch skupín:

- **Abstraktné modely** sú založené na nejakom **formálnom spôsobe popisu** modelovaného systému a tento formálny aparát je aj prostredkom pre skúmanie vlastností takéhoto modelu. Príkladom môže byť entitno-relačný model bázy dát (ER model). Experimentovanie s modelom (skúmanie jeho vlastností) vyžaduje znalosť použitého formálneho aparátu a aj znalosť skutočnej fyzikálnej odozvy modelovaného systému. Experimentovanie s takýmto modelom je experimentovaním v predstavách a na papieri.
- **Virtuálne modely** sú rozšírené abstraktné modely, ktoré prezentujú svoje fyzikálne vlastnosti virtuálne - nie v reálnom, ale vo virtuálnom prostredí vytvorenom počítačom. Napr. programovo implementovaný entitno-relačný model bázy dát, s ktorým je možné experimentovať (skúmať vlastnosti namodelovanej bázy dát), aj keď fyzický báza dát neexistuje.
- **Reálne modely** sú z nejakého hľadiska zjednodušené podoby systému funkčné v tejto zjednodušenej forme v reálnych podmienkach. Napr. model informačného systému, v ktorom sú implementované len niektoré z požadovaných služieb, alebo je implementované len používateľské rozhranie.

Pod **prototypom** rozumieme reálne funkčné verzie systému, ktorá nemá implementované všetky vlastnosti konečného produktu a slúži na otestovanie niektorých vybraných vlastností navrhovaného systému.

Prototyp, ktorý má v sebe zahrnuté riešenie všetkých požadovaných vlastností systému (alebo aspoň autori systému si to myslia) a je určený predovšetkým na testovanie (odhalenie skrytých chýb a chýbajúcich vlastností), sa označuje ako **B-verzia systému**.

Prototyp **programového systému** je z pohľadu modelovania vlastne spojením reálneho a virtuálneho modelu systému. Podľa toho budeme ďalej uvažovať aj o reálnej a virtuálnej zložke prototypu:

- **Reálna zložka prototypu** zabezpečuje prezentáciu tých vlastností budúceho systému, ktoré budú zhodné s vlastnosťami cieľového systému. T.j. sú to vlastnosti systému, ktoré už sú v prototype plne implementované.

- **Virtuálna zložka prototypu** prezentuje tie vlastnosti systému, ktoré sú buď nejakým spôsobom simulované (neprázdná virtuálna zložka) alebo ignorované (prázdná virtuálna zložka).

Čím nižší je podiel virtuálnej zložky v modele, tým viac sa prototyp bliží k **B-verzie systému**. Od podielu simulovaných a ignorovaných vlastností systému v prototipe závisí **hodinovnosť prototypu**. Ak napr. v prototipe používateľského rozhrania sú prezentované len vizuálne vlastnosti a spôsob riadenia interakcie používateľ-systém, ale ďalšie vlastnosti cieľového systému sú ignorované (napr. prístupové práva k jednotlivým službám, údajom, časová odozva pri operáciach s bázou dát, reakcia na chybové stavy a pod.), prototyp je mälo **spôsahlivý** pre posúdenie vlastností navrhovaného systému. Ak tento prototyp bude simulať aj časovú odozvu (odhadovanú podľa vlastnosti dátového modelu a implementačného prostredia), experimentovanie s ním môže ukázať absolútnu nevhodnosť niektorých obrazoviek pre danú štruktúru dátového modelu.

2.3. Modelované aspekty softvérových systémov - oblasti modelovania

Na softvérový systém a aj na proces jeho vývoja, zavedenia a používania – t.j. procesy spojené so **životným cyklom softvéru**, sú kladené rozličné **kvantitatívne a kvalitatívne požiadavky**. Napr. systém musí byť zrealizovaný v určitom čase, v rámci určitého rozpočtu, musí poskytovať požadované služby, sprístupňovať alebo spracovať požadované informácie, zabezpečovať riadenie iných systémov a pod. Abstrakciu týchto požiadaviek vznikajú požadované **abstraktné pohľady** (na systém a procesy spojené s jeho životným cyklom). Napr. pohľad na realizáciu systému z hľadiska času (časový plán – sieťový model úloh), z hľadiska nákladov (model pre cenový odhad), pohľad na vstupné, výstupné a interné informácie v systéme, ich štruktúru a vzájomné väzby (dátový model), pohľad na realizované procesy a dátové toky (procesný model) atď.

Základné oblasti modelovania súvisiace s činnosťami počas životného cyklu softvérového systému sú:

1. **Životný cyklus softvérového systému** – jeho vnútorná štruktúra (etapy, procesy, kroky) a vzájomné väzby
2. **riadenie softvérového projektu** – riadenie jednotlivých etáp a dielčích krovok životného cyklu – časové a personálne plány súvisiacich prác
3. **organizačná štruktúra realizačného tímu** – kompetencie a vzájomná komunikácia v tíme
4. **kvantitatívne vlastnosti softvérového systému** – rozsah a cena programov, rýchlosť realizácie služieb, mohutnosť báz dát, mohutnosť prenášaných dát a pod.
5. **kvalitatívne vlastnosti softvérového systému**:
 - **oblasť procesov** (činnosti, aktivít, funkcií, služieb)
 - **oblasť dát** (údajových štruktúr, databáz, tried a objektov)
 - **oblasť riadenia** (časové a iné súvislosti, reakcie na udalosti)
 - **oblasť komunikácie** (medzisystémová a vnútrosystémová, dávková a interaktívna – používateľské rozhranie)
 - **oblasť architektúry systému** (softvérovej a hardvérovej, modulárna štruktúra, alokácia procesov na procesoroch a báz dát na databázových servroch a pod.)

Uvedeným oblastiam prislúchajú abstraktné pohľady na systém (procesný, dátový, riadiaci,...) orientujúce sa práve len na tie vlastnosti systému, ktoré s danou oblasťou súvisia. z toho by sa mohlo zdáť, že vytvárané abstraktné modely budú navzájom nezávislé. Nie je tomu tak, existujú súvislosti medzi modelmi (vid. kap.5).

Rôzne pohľady na systém (rôzne oblasti modelovania vlastností systému) umožňujú vytvárať modely paralelne - v uvedenom chápaní nezávisle. Vzhľadom na tento možný paralelný vývoj sa uvádzajú 3 hlavné smery: dátový (údajový), funkčný (procesný) a riadiaci a im zodpovedajúce ortogonálne modely:

- dátový model (DM – Data Model)
- funkčný model (FM – Function Model)
- riadiaci model (CM – Control Model)

V rámci 3-dimenzionálneho pohľadu DM-FM-CM môžeme uvažovať rozšírené modely DM, FM, CM:

- v FM uvažovať okrem procesov, funkcii a služieb aj funkčnú architektúru systému (napr. moduláru štruktúru aplikácie)
- v DM uvažovať okrem štruktúry dát a ich vzájomných väzieb aj dátovú architektúru (napr. rozmiestnenie dát na databázových servoch)
- v CM uvažovať aj oblasť komunikácie, vrátane používateľského rozhrania.

2.4. Súvislosť modelov a prototypov

Súvislosť medzi prototypmi a modelmi systému sa prejavuje dvojako:

- prototyp je spojenie reálneho a virtuálneho modelu systému (t.j. prototyp je model systému s neprázdnou reálnou zložkou, virtuálna zložka môže byť prázdna)
- jednotlivé zložky prototypu (reálna aj virtuálna) môžu byť generované na základe zodpovedajúceho abstraktného modelu (jeho formálnej špecifikácie).

Vzťah medzi abstraktným modelom systému a odpovedajúcim prototypom je vzťahom matematického modelu a jeho programovej implementácie.

Ak je prototyp vytvorený na základe jediného abstraktného modelu (napr. dátového), budeme ho ďalej nazývať **analytickým prototypom**.

Analytický prototyp je z pohľadu modelovania reálny model alebo z pohľadu prototypovania prototyp s prázdnou virtuálnou zložkou. Analytický prototyp zodpovedajúci danému abstraktnému modelu je možné generovať z príslušnej formálnej špecifikácie popisujúcej abstraktný model.

Ak je prototyp vytvorený na základe viacerých abstraktných modelov (nemusia byť úplné) predstavujúcich všetky typy sledovaných vlastností systému, budeme ho nazývať **komplexným prototypom**.

Komplexný prototyp je prototyp s neprázdnou virtuálnou zložkou. Reálna zložka prezentuje reálne určité množinu vlastností cieľového systému a virtuálna zložka inú množinu vlastností simuluje. Napr. komplexný prototyp používateľského rozhrania (PR), na rozdiel od analytického, by mal prezentovať okrem vizuálnych vlastností a predpokladanú časovú odozu (simulať prístup k báze dát).

Komplexné prototypy zohľadňujú všetky tri základné pohľady na systém: dátový, funkčný, používateľský. Pritom niektoré vlastnosti môžu byť simulované (virtuálna zložka), iné prezentované realisticky (reálna zložka).

Analytický prototyp zohľadňuje jeden pohľad na systém. Pre zostavenie analytického prototypu používateľského rozhrania vystačíme s neúplným dátovým modelom (podstatné sú dátové elementy, nie entity a relácie) a s neúplným funkčným modelom (len interaktívne procesy, ich vstupy a výstupy). Pre doplnenie virtuálnych zložiek do komplexného prototypu PR je potrebné poznáť úplný relačný dátový model, kvantitatívne odhady veľkosti tabuľiek a aj vlastnosti implementačného prostredia. Na základe týchto parametrov je možné odhadnúť časové odozy prístupov k údajom v danej skupine entít a tento prístup simulať časovým oneskorením reakcií v prototype.

VÝZNAM ABSTRAKTNÝCH MODELOV PROGRAMOVÝCH SYSTÉMOV

Pre analýzu a návrh systémov majú v súčasnosti najväčší význam grafické a tabuľkové reprezentácie abstraktných modelov - **grafické nástroje**. Zrejmým dôvodom sú prednosti viacerozmernej grafickej informácie oproti jednorozmernosti textových špecifikácií. Použitie abstraktných modelov a im odpovedajúcich nástrojov v životnom cykle programových systémov má nasledujúce výhody:

- stručnosť, prehľadnosť a zrozumiteľnosť pre profesionálov (sú vhodné pre systémovú dokumentáciu)
- umožňujú automatické generovanie zodpovedajúcich častí systému
- umožňujú detekciu formálnych chýb v návrhu z pohľadu jednoho modelu
- umožňujú detekciu formálnych chýb v návrhu porovnaním závislostí medzi modelmi (ak existujú)
- umožňujú udržiavať konzistenciu medzi dokumentáciou systému a jeho implementáciou po zmenách, opravách a rozšíreniach
- zo súboru detailne rozpracovaných modelov, ktoré pokrývajú všetky dôležité vlastnosti systému, je možné generovať β-verziu systému

Nevýhody abstraktných modelov:

- sú málo zrozumiteľné budúcim používateľom systému - vyžadujú znalosť formálneho aparátu, ktorý bol pre zostavenie modelov použitý (ER-modely - ER diagramy, konečnostavové automaty - stavové diagramy a pod.)
- skompletovanie niektorých abstraktných modelov u rozsiahlych systémov je časovo náročné

VÝZNAM PROTOTYPOV PROGRAMOVÝCH SYSTÉMOV

Použitie prototypov v životnom cykle programových systémov má nasledovné výhody:

- sú zrozumiteľné koncovým používateľom systému - prototypy sú bližšie koncovému používateľovi ako abstraktné modely
- umožňujú nájsť logické chyby a nedostatky v návrhu pri testovaní používateľom (chyby sú detektovateľné rýchlejšie, ako keby sa malo čakať až na β-verziu)

- ak majú súvislosť s nejakým abstraktným modelom, sú automaticky generovateľné - môžu byť rýchle k dispozícii

Nevýhody vytvárania prototypov:

- veľmi rýchle vytvorený prototyp vytvára u používateľa (resp. zadávateľa projektu) dojem, že aj ďalšie práce musia ísť rovnako rýchle a bez problémov - t.j. ak je hotový prototyp, na vytvorenie konečnej verzie už nebudú takmer žiadne náklady
- používateľom odsúhlásený prototyp (napr. demonštrujúci len vizuálne dialógové vlastnosti budúceho systému) môže v konečnom dôsledku vykazovať značne odlišné parametre ako koncový systém (napr. z hľadiska rýchlosťi odozvy, ak model používateľského rozhrania nedostatočne reálne simuloval čas potrebný na sprístupnenie nejakých informácií z bázy dát - tento čas závisí aj od vlastnosti dátového modelu.)

3. Modelovanie životného cyklu programových systémov

Programové systémy všeobecne, tak ako aj všetky technické systémy, prechádzajú od svojho zrodu v podobe predstáv až po reálne používané systémy niekoľkými fázami (etapami). V predprojektových a projektových fázach sú systémy vo forme rôznych dokumentov, modelov a prototypov. Vo fáze testovania a používania už majú podobu programov vykonateľných na požadovanom výpočtovom systéme. Výpočtovým systémom budeme rozumieť určitú konfiguráciu technických a základných (systémových) programových prostriedkov.

Všeobecne uznávané sú 4 fázy, ktorými programové systémy prechádzajú:

- analýza** (analysis) - analýza vlastností existujúceho systému, špecifikácia požiadaviek na nový systém
- návrh** (design) - návrh štruktúry a vlastností nového systému
- implementácia** - implementácia navrhovaného systému v konkrétnom technickom a programovom prostredí
- používanie a údržba**

Pokiaľ systém úplne nezanikne, tieto štyri fázy sa cyklicky opakujú pri inovácií (opravách, rozšírení, modernizácii, vylepšovaní,...) existujúceho systému. Postupnosť opakujúcich sa fáz systému tvorí cyklus, ktorý nazývame **životný cyklus softvéru** (SwLC - Software Life Cycle).

Jednotlivé etapy životného cyklu programového systému je možné rozčleniť detailnejšie nasledovne:

Predprojektové etapy	Štúdia realizovateľnosti systému <ul style="list-style-type: none"> analýza existujúceho systému specifikácia problému analýza možností riešenia predpoklady riešenia (časové, finančné, personálne, iné) plán projektu
Projektové etapy	Hĺbková analýza existujúceho systému <ul style="list-style-type: none"> definovanie hraníc systému specifikácia nedostatkov existujúceho systému specifikácia používateľských požiadaviek na nový systém Návrh nového systému (tzv. programovanie vo veľkom) <ul style="list-style-type: none"> návrh architektúry systému detaljný návrh

Implementácia (tzv. programovanie v malom) Skompletovanie dokumentácia Testovanie Školenie personálu Konfigurácia Inštalácia Skúšobná prevádzka	Prevádzka Údržba Testovanie Modifikácie Správa verzii Nové inštalácie Evidencia nových používateľských požiadaviek
Etapy používania systému	

Identifikácia etáp životného cyklu (ich ohraničenie a presná špecifikácia zdrojov, prostriedkov a cieľov) a ich návaznosti umožňuje:

- nájsť postupy pre zvýšenie produktivity vývojových prác
- odhadnúť časové nároky na jednotlivé etapy
- odhadnúť celkovú cenu projektu
- využiť nástroje a systémy pre počítačom podporované softvérové inžinierstvo (CASE - Computer Aided Software Engineering)

Nástroj (prostriedok, tool) je grafický, textový alebo iný formálny alebo semiformálny prostriedok pre modelovanie nejakého aspektu programového systému.

Nástroje, ktoré metódy používajú, majú rôznu formu. Obyčajne sú:

- grafické - vychádzajúce z rozličných grafov a diagramov (napr. vývojový diagram, entitno-relačný diagram)
- tabuľkové (napr. matica udalostí)
- textové (napr. popis algoritmu pomocou formálneho jazyka, popis štruktúry dát pomocou Backus-Naurovej formy)

Metóda (method) je technika (procedúra, postup) pre vykonanie nejakej časti životného cyklu (SwLC), používajúca nejaké nástroje. Napr. známa je Chennova metóda návrhu relačnej štruktúry dát, Jacksonova metóda návrhu štruktúry dát a algoritmov, metóda modulárneho programovania, metóda objektovo-orientovaného programovania a iné. Každá z týchto metód má okrem nástrojov definovaný aj cieľ. Nástroje slúžia na dosiahnutie tohto cieľa v určitej fáze životného cyklu programu.

Metodológia je súbor metód podporujúcich na základe spoločnej filozofie viac etáp životného cyklu. Medzi známe metodológie patrí napr. YSM (Yourdonova metodológia struktúrovanéj analýzy a návrhu systémov), OMT (Rumbaughova objektovo-orientovaná metodológia analýzy a návrhu systémov). V anglickom texte sa tieto postupy označujú ako metódy (methods).

Vzhľadom na význam slov metóda a metodológia je zaužívané v domácich odborných publikáciách oba pojmy v softvérovom inžinierstve rozlišovať uvedeným spôsobom.

Životný cyklus programových systémov je charakterizovaný nielen etapami (fázami), ich obsahom, ale aj:

- vzájomnou následnosťou fáz
- podmienkami ukončenia fázy (prechodu od jednej fázy k druhej)
- použitými metódami, nástrojmi a metodológiami

Tieto vlastnosti životného cyklu definuje tzv. **model životného cyklu**. Jednotlivé používané modely životného cyklu sú výsledkom nahromadených praktických skúseností z vývoja rozsiahlych programových systémov v organizáciách, ktoré sa takýmto vývojom profesionálne zaoberejú. Nie všetky modely životného cyklu sú vhodné pre každý programový systém. Volba modelu životného cyklu je závislá nielen od vlastnosti systému (jeho rozsahu, zložitosti, požiadaviek na bezpečnosť a pod.), ale aj od vlastnosti subjektov, ktoré sa na životnom cykle systému podieľajú (organizácia, pracovný tím, časové a finančné kapacity a pod.).

3.1. Modely životného cyklu programových systémov

Modely SwLC definujú návaznosť jednotlivých etáp SwLC. Sú to abstraktné modely, ktoré slúžia pre plánovanie, riadenie a realizáciu:

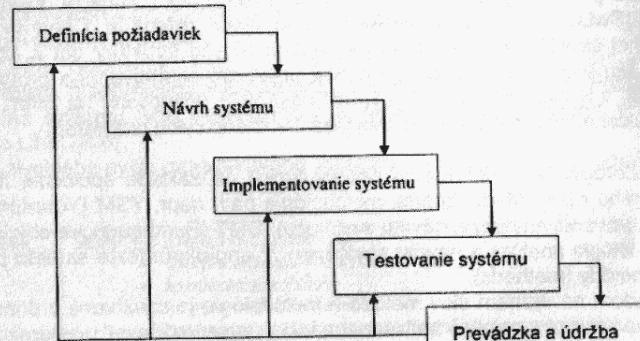
- predprojektových a projektových prác na programovom systéme
- správy a údržby programového systému

V súčasnosti medzi najčastejšie používané modely životného cyklu programového systému patria:

- vodopádový model
- prototypovací model
- model prieskumník
- špirálový model (Bohm, [5])

3.1.1. Vodopádový model

Schématické znázornenie:



Obr. 3.1 Vodopádový model životného cyklu

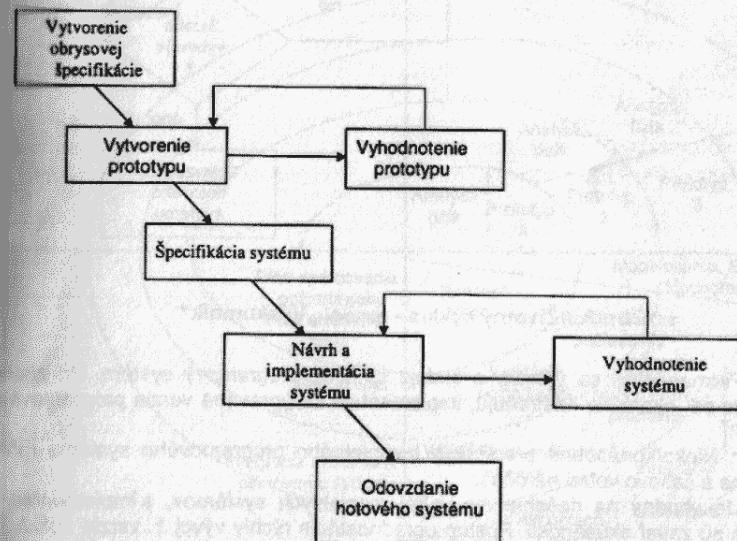
Výhody: Je presne definované ukončenie jednej etapy a začiatok ďalšej. Práce v nasledujúcej etape vychádzajú z výsledkov predchádzajúcej etapy, ktoré sa považujú za 100% správne.

Nevýhody: Ak výsledok niektoréj z etáp je nesprávny, aj výsledný programový systém nebude zodpovedať pôvodným požiadavkám a náklady na opravu budú tým vyššie, čím skôr došlo k chybe. Ďalšou nevýhodou tohto modelu je, že úplné ukončenie jednej etapy môže trvať veľmi dlho, a začatie ďalšej etapy musí byť odkladané (napr. analýza rozsiahleho systému môže trvať niekoľko mesiacov a nemôže byť urobený návrh systému).

Použitie: Vodopádový model je výhodné použiť iba pri riešení často opakovanych podobných programových systémov, ktoré nie sú veľmi rozsiahle.

3.1.2. Prototypovací model

Schématické znázornenie:



Obr. 3.2 Prototypovací model životného cyklu

Výhody: Hlavnou výhodou tohto modelu je, že poskytuje ešte pre dokončením celého systému jeho v nejakom smere zjednodušenú "verziu" - prototyp. Na tomto prototypu je možné overiť niektoré vlastnosti budúceho systému a včas zistiť pripadné nedostatky a chyby v analýze alebo návrhu.

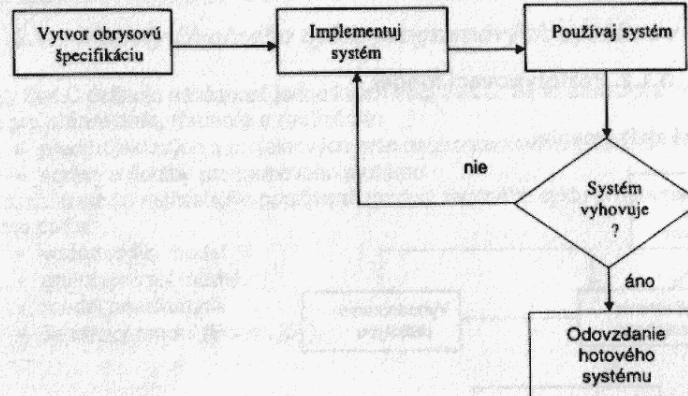
Nevýhody: Prototyp systému je možné spravidla zostaviť veľmi rýchle. To na jednej strane umožňuje ešte pred konečnou implementáciou systému otestovať jeho niektoré

vlastnosti a splnenie niektorých používateľských požiadaviek. Na druhej strane ale rýchla výroba prototypu môže vytvárať u zákazníka falošnú ilúziu rýchleho dokončenia celého systému. Pri prototypovaní môže byť problémom aj absencia vhodnej analytickej dokumentácie projektovaného systému na vyšej úrovni abstrakcie.

Použitie: Prototypovací model životného cyklu je vhodný pre tvorbu menej rozsiahlych systémov.

3.1.3. Model výskumník

Schématické znázornenie:



Obr. 3.3 Životný cyklus - model „Výskumník“

Výhody: Permanentne sa vytvára a teste funkčný programový systém. Pri zistení nedostatkov sa okamžite odstraňujú, implementuje sa upravená verzia programového systému.

Nevýhody: Niekoľkonásobná prerábanie kompletného programového systému môže byť finančne a časovo veľmi náročné.

Použitie: Je vhodný na riešenie nie veľmi rozsiahlych systémov, s implementáciou ktorých nie sú zataľ skúsenosti. Postup uprednostňuje rýchly vývoj 1. verzie aplikácie. Do konečnej podoby podľa požiadaviek používateľa sa vyvíjaný systém dostane cyklickými úpravami, dolaďovaním a modifikovaním pôvodnej verzie.

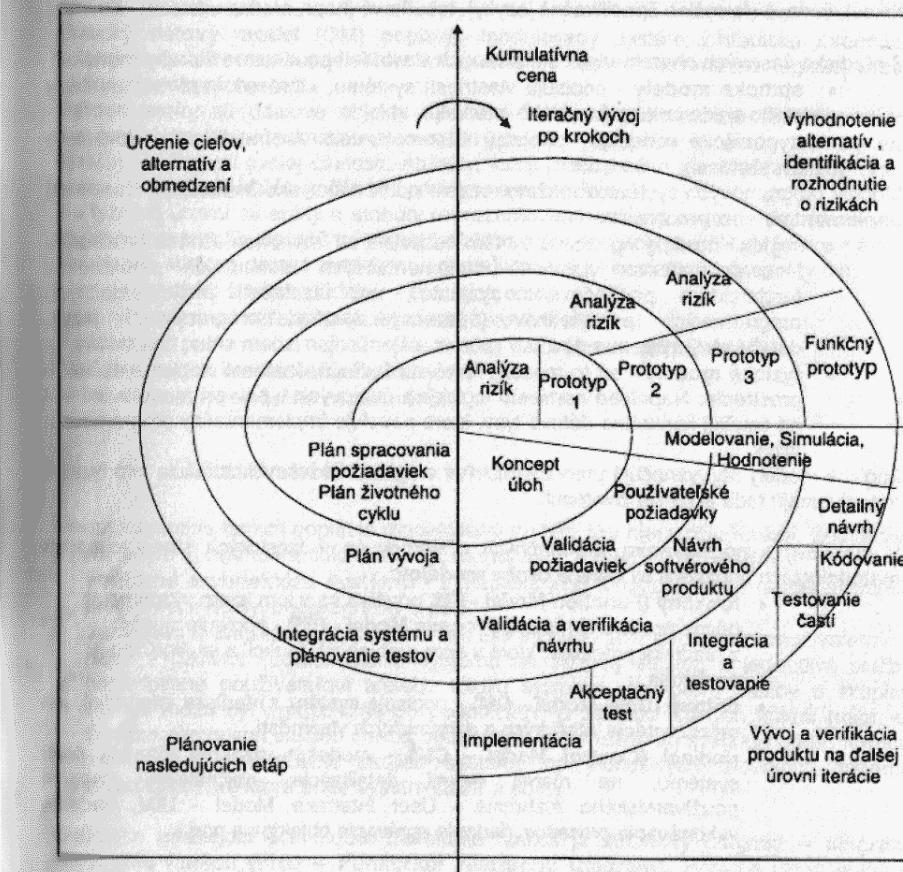
3.1.4. Špirálový model

Výhody: Vývoj programového systému prebieha po špirále. Cyklicky prechádza všetkými etapami, pritom sa začína s malou časťou systému (jadrom, pilotným projektom) a tá sa postupne rozširuje. Keď je ukončená jedna časť systému, prechádza sa k ďalšej.

Nevýhody: Postupné rozširovanie aplikácie o ďalšie časti vyžaduje vhodný výber pilotného projektu, dôslednú modularitu a predvídanie vlastností budúcich modulov. Pratolože v každom cykle špirálového vývoja je potrebný návrat k predošej verzii systému, je potrebné klášť vysoký dôraz na neustálu aktuálnosť projektovej dokumentácie - otázku jej skompletovania nie je možné odkladať na dobu "po ukončení programátorovských prác" (táto zdaničivá nevýhoda väčšieho úsilia a množstava práce v úvodných cykloch riešenia projektu sa zúročí neskôr ako nesporná výhoda!).

Použitie: Je vhodný na riešenie rozsiahlejších systémov, s implementáciou ktorých nie sú skúsenosti.

Behématické znázornenie:



Obr. 3.4 Špirálový životný cyklus

4. Modely pre analýzu a návrh programových systémov

Štruktúrovanosť pohľadu na analyzovaný a navrhovaný systém dovoluje abstrahovať od nepodstatných detailov na danej úrovni analytických a návrhárskych prác a súčasne vytvoriť predpoklady pre potrebnú detailizáciu v ohraničených, presne špecifikovaných rámcoch - štruktúrach. Štruktúra ako abstrakcia sa môže týkať rôznych aspektov existujúceho alebo projektovaného systému. Z vhodne popísaných štruktúr je možné zostaviť abstraktné modely systémov vhodné na experimentovanie a posudzovanie vlastností budúceho systému ešte pred jeho (spravidla nákladou) implementáciou. Pre zostavenie modelov sa používajú rôzne **nástroje**: grafové (napr. diagram dátových tokov), textové (formálne špecifikačné jazyky), tabuľkové (napr. matica udalostí).

Z hľadiska časových charakteristik sledovaných vlastností používame 2 druhy modelov:

- **statické modely** - popisujú vlastnosti systému, ktoré sú nezávislé od času alebo sledovaného časového intervalu
- **dynamické modely** - popisujú časovo závislé vlastnosti alebo správanie systému.

Modely programových systémov môžeme rozdeliť podľa toho, ako zohľadňujú **vlastnosti implementačného prostredia**, na:

- **logické modely** - popisujú systém nezávisle od implementačného prostredia. Nezohľadňujú sa vlastnosti implementačných technických prostriedkov (architektúra počítačového systému), ani vlastnosti implementačných programových prostriedkov (operačný systém, programovací jazyk, databázový systém a pod.).
- **fyzické modely** - sú to modely, ktoré zohľadňujú vlastnosti implementačného prostredia. Napríklad namiesto logických údajových typov pri modelovaní dát sa použijú konkrétné dátové typy, ktoré povoluje implementačný programovací jazyk.

Logické modely sa vyznačujú prenositeľnosťou a vyššou úrovňou abstrakcie ako fyzické modely, majú teda aj dlhšiu životnosť.

V súvislosti s modelovaním informačných systémov sa v klasických štruktúrovaných metodológiách používajú tri hlavné druhy modelov:

- **funkčný (Function Model - FM)**, používa sa v tom istom význame aj názov **procesný model**, **Process Model - PM**) - popisuje systém z hľadiska procesov, ktoré v ňom prebiehajú, funkcií a služieb, ktoré poskytuje.
- **dátový (Data Model - DM)** - popisuje systém z hľadiska informácií, ich reprezentácie, statických a dynamických vlastností.
- **riadiaci (Control Model - CM)** - modeluje riadenie rôznych častí systému, na rôznej úrovni detailizácie, najčastejšie riadenie používateľského rozhrania - User Interface Model - UIM, riadenie vykonávania procesov, riadenie správanie objektov a pod.).

Hoci uvedené modely je možné vytvárať pri analýze a syntéze systému paralelne, nie sú to nezávislé modely. **Súvislosti** medzi nimi sú veľmi význačné na to, aby ich bolo možné zanedbať na úrovni logického aj fyzického modelu. Naopak využitím týchto súvisostí je možné odhaliť nekonzistencie alebo chyby v analýze alebo syntéze systému (viď. kap. 5).

4.1. Modelovanie statických vlastností programových systémov

Statické modely popisujú tie vlastnosti modelovaného systému, ktoré sú nemenné v čase alebo určitom časovom intervale. Popisujú štruktúru a vnútorné väzby v systéme a pohľadu funkcií aj dát. V klasickej štruktúrovanej analýze sa vytvárajú oba pohľady spoločne izolované, neskôr sa hľadajú vzájomné súvislosti. Pri objektovej analýze a návrhu sa uvažuje štruktúra systému na báze objektov – abstrakcií, ktoré uzavárajú data a súvisiace funkcie do jedného štrukturálneho prvku.

Statický funkčný model (FM) popisuje systém z hľadiska jeho štruktúry na báze systémov, podsystémov a funkcií – pre tento účel sa používa modelovanie hierarchie funkcií.

Statický dátový model (DM) popisuje modelovaný systém z hľadiska informácií, s ktorými systém manipuluje. Tieto informácie sú reprezentované na logickej úrovni výskytom dátových entít.

Dátové entity sú údajové objekty systému, ktoré sú jednoznačne identifikovateľné a charakterizovateľné pomocou elementárnych dát - **atribútov** a všetky atribúty entít systém využíva pri svojej činnosti. Atribúty, ktoré jednoznačne identifikujú každý výskyt entít, sa nazývajú **kľúče** (môžu byť primárne a sekundárne).

Na fyzickej úrovni sú entity a atribúty reprezentované rozličnými údajovými štruktúrami vo vnútornej pamäti (operačnej pamäti) a/alebo na vonkajších pamätiach (disky, pásky). Používané statické dátové modely sa orientujú najmä na nasledujúce aspekty dát:

- statická štruktúra údajov – dátových entít
- vzájomné vzťahy (relácie) medzi dátovými entitami

V súčasnosti patria medzi najbežnejšie metódy dátového modelovania:

- metóda modelovania štruktúry dát
- metódy relačného modelovania

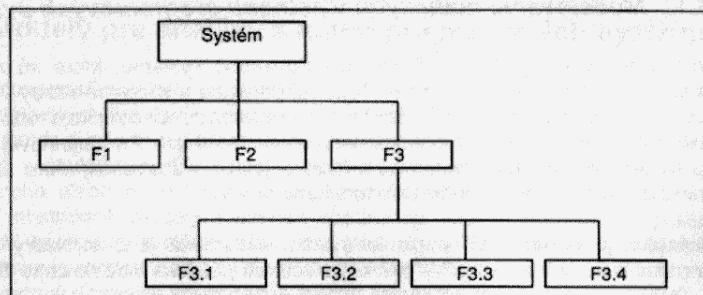
4.1.1. Model hierarchie funkcií

Model hierarchie funkcií popisuje modelovaný systém ako hierarchiu funkcií. Sémantika funkcií a tým aj celého modelu môže byť rôzna, napr. model môže popisovať:

- **vnútornú architektúru systému** - z hľadiska jeho štruktúralizácie na podsystémy, programy, moduly, podprogramy a pod.
- **vonkajšiu architektúru systému** – tak ako sa bude javiť budúcemu používateľovi – napr. z hľadiska štrukturalizácie systému na skupiny služieb a jednotlivé služby poskytované používateľovi a/alebo okolia systému. Jednotlivé služby a skupiny služieb môžu byť zgrupované do podmnožín na základe rôznych kritérií (napr. v súhlase s ponukou služieb danou používateľským rozhraním systému (napr. formou menu), podľa distribúcie služieb systému na jednotlivých úrovniach riadenia organizácie, pre ktorú bude systém slúžiť a pod.)

Grafovým nástrojom pre model hierarchie funkcií je stromový diagram – **diagram hierarchie funkcií (FHD – Funktion Hierarchy Diagram)**. Poradie uzlov grafu – funkcií na jednej úrovni dekompozície nemá žiadnený vzťah k poradiu vykopávania týchto funkcií.





Obr. 4.1 Diagram hierarchie funkcií

Reprezentácia FHD používa prvky podľa nasledujúcej tabuľky:

PRVOK	NÁZOV	MODELUJE
[]	Funkcia	Koreňový uzol predstavuje systém, podsystém, program, modul, podprogram, množinu služieb alebo službu. Nekoncový uzol modeluje množinu funkcií (služieb). Koncový (listový) uzol stromu modeluje elementárnu funkciu alebo službu.
[—]	Dekompozícia funkcie	V smere od koreňa k listom predstavujú hrany smer dekompozície funkcie.

Rozsiahle systémy môžu mať veľký počet funkcií. Pre prehľadnosť diagramu sa v tomto prípade používajú ako listové položky stromu aj uzly predstavujúce podmnožiny služieb. K nim existuje dekompozícia vo forme ďalšieho stromu FHD. Pri identifikácii funkcií sa používa desatinné kódovanie – stromová závislosť je jednoznačne definovaná aj týmto kódom. Koncový uzol, ktorý má svoju dekompozíciu v inom FHD, má zhodný identifikátor, ako je identifikátor koreňového uzla stromu predstavujúceho dekompozíciu. Diagram hierarchie funkcií zohľadňujúci len interaktívne dostupné funkcie systému modeluje aj používateľské rozhranie a môže byť súčasťou dynamického modelu komunikácie pomocou ponuky (menu) služieb (vid. kap. 4.2.3). Podobne diagram hierarchie funkcií modelujúci vnútornú architektúru systému (podsystémy, programy, moduly, podprogramy a pod.) je jadrom dynamického modelu modulárnej štruktúry, ktorý naviac modeluje dátové a riadiace toky medzi modulmi a podprogramami (vid. kap. 4.2.2).

4.1.2. Hierarchický model štruktúry dát

Pre reprezentáciu hierarchickej štruktúry dát sa používa diagram dátových štruktúr (DSD - Data Structure Diagram). Tento model súvisí aj s metódou JSP (Jackson Structured Programming), ktorá je založená na návrhu štruktúry algoritmov podľa štruktúry dát, s ktorými algoritmus manipuluje (JSD, vid. kap. 4.2.2).

Diagram DSD je stromový diagram, jeho prvky tvoria:

- uzly: koncové (dátové atribúty), nekoncové (dátové entity):
 - povinné dátá (vyjadrenie sekvencie dát)
 - voliteľné dátá (vyjadrenie selekcie v dátach)
 - opakujúce sa dátá (vyjadrenie iterácie v dátach)
- hrany: dekompozícia štruktúry dátovej entity (na sekvencie, selekcie, iterácie dát)

Koreňom stromu je uzol zodpovedajúci modelovanému dátovému objektu. Diagram štruktúry modeluje detailne jednotlivé povinné, nepovinné a opakujúce sa časti dátového objektu (sekvencie, selekcie, iterácie).

Reprezentácia DSD používa také isté grafické prvky ako diagram štruktúry algoritmu podľa metódy JSP.

PRVOK	NÁZOV	MODELUJE
[]	Povinná zložka štruktúrovaného dátového typu	Štruktúrovaný dátový typ (v prípade nekoncového uzla) - logické alebo fyzické zoskupenie štruktúrovaných dátových typov alebo elementárnych dátových typov predstavujúcich: štruktúru formulárov, zostav, prehľadov, obrazoviek,... Elementárny dátový typ (v prípade listového uzla stromu) zodpovedá: jednotlivej rubrike, položke...
[○]	Nepovinná zložka	Zoskupenie údajov alebo jednotlivý údaj, ktorý môže zostať nevyplnený, alebo môže chýbať: nepovinne vyplňiteľná rubrika alebo skupina rubriek formulára.
[.]	Opakujúca sa zložka	Opakujúca sa zložka dát rovnakého typu.
[—]	Dekompozícia údajového typu	V smere zhora nadol predstavujú hrany smer dekompozície štruktúrovaného údajového typu na zložky.

Metóda modelovania štruktúry dát sa používa v CASE systémoch strednej úrovne (middle CASE) - pre modelovanie štruktúry dátových tokov (zodpovedajúcich napr. obrazovkovým formulárom, tlačeným zostavám a pod.), prípadne aj modelovanie štruktúry dátových pamäti. T.j. pre dekompozíciu dátových pamäti a dátových tokov v diagramoch dátových tokov (DFD, vid. kap. 4.2.1, 5) môžu byť výhodne použité diagramy dátových štruktúr DSD.

Často sa štruktúra dát reprezentuje nie v grafickej, ale v textovej podobe (hlavne # úspomínkových dôvodov) pomocou Backus-Naurovej formy (BNF). BNF je metajazyk pre popis syntaxe jazykov. Dekompozícia sa vyjadruje pomocou gramatických (produkčných) pravidiel s tvarom:

$$A ::= B_1 B_2 B_3 \dots B_n$$

kde A je dekomponovaná dátová entita (syntaktická kategória, neterminálny symbol) a B_i sú buď neterminálne alebo terminálne symboly (dátové elementy, ktoré sa ďalej

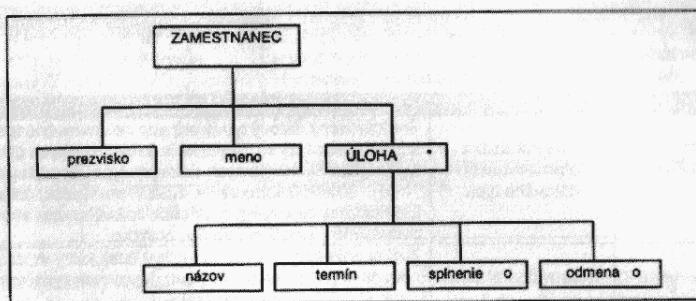
nedekomponujú) s definovanou početnosťou výskytu, napr. pomocou zátvoriek a znakov +, * nasledovne:

- výskyt práve jedenkrát: [B]
- výskyt nula alebo jedenkrát (voliteľná entita): [B]^0
- výskyt nula alebo viackrát (iterácia): [B]^*
- výskyt jeden alebo viackrát (iterácia): [B]^+

Terminály a neterminály sú v pravidlach rozlišené napr. veľkými a malými písmenami (názvy neterminálov sú z veľkých písmen, terminálov z malých).

Ďalej nasleduje príklad štruktúry dátového objektu - výstupnej zostavy v tvare BNF aj v tvare zodpovedajúceho diagramu dátových štruktúr (DSD):

```
ZAMESTNANEC ::= priezvisko meno [ÚLOHA]*  
ÚLOHA    ::= názov termín [splnenie] [odmena]
```



• Obr. 4.2 Jacksonov diagram štruktúry dát

4.1.3. Entitno-relačný model dát

Entitno-relačný model modeluje všetky podstatné údajové entity (entity, údajové objekty) v systéme, ich štruktúru, obsah a ich vzájomné vzťahy. Hlavný problém vytvorenia dátového modelu (a úlohou dátového modelovania) je rozpoznanie entít a vzťahov medzi nimi.

Grafická reprezentácia entitno-relačného modelu je **entitno-relačný diagram (ERD - Entity Relationship Diagram)**. Prvky modelu tvoria entity, relácie, atribúty, kľúče. Primárne kľúče slúžia na jednoznačnú identifikáciu výskytov entít. Pre implementáciu relačného modelu je prirodzená implementácia pomocou relačného databázového systému, v ktorom entite zodpovedá tabuľka, atribúty sú stĺpce a relácia sa realizuje pomocou tzv. *cudzích kľúčov*. Cudzie kľúče vytvárajú referenciu na ďalšie výskupy tej istej alebo inej entity.

Pre relačné dátové modelovanie existuje viacero reprezentácií entitno-relačných diagramov a im zodpovedajúcich metód. Rozšírené je používanie ERD v notácii podľa Chenna, Yourdona, metodológie SSADM (System Structured Analysis/Design Method)

a Oracle®Method. Diagramy sa v jednotlivých metódach líšia v podstate len vizuálnou reprezentáciou a možnosťami pre modelovanie niektorých špeciálnych relácií. V metóde podľa Chenna je ERD sietový diagram, ktorého prvky tvoria:

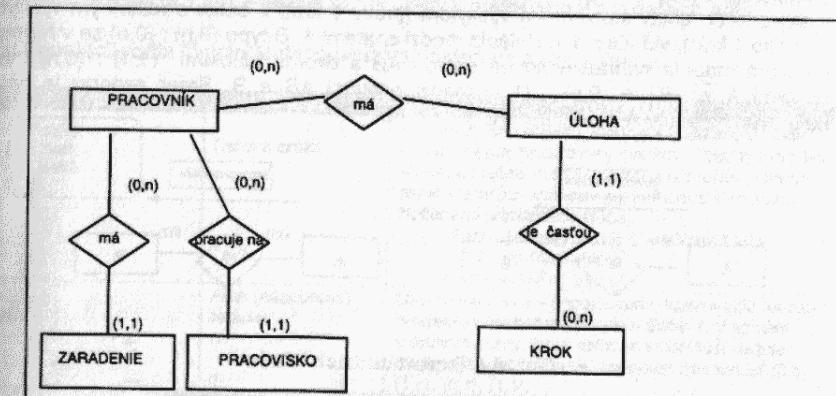
- uzly: dátové objekty (entity) a relácie
- hrany: spojenie entít prostredníctvom relácií

- Chennova notácia ERD používa nasledujúce uzly a hrany:

PRVOK	NÁZOV	MODELUJE
	Dátová entita	Typ údajového objektu, ktorý je z nejakého dôvodu významný pre modelovaný systém. Objekty tohto typu sú jednoznačne identifikovateľné pomocou jedného alebo viacerých atribútov (t.j. jednoduchého alebo zloženého primárneho kľúča).
	Relácia	Vzťah medzi dvomi alebo viacerými entitami kvantifikovaný počtom výskytov entít v tomto vzťahu (je možná aj rekúzívna relácia na jednej entite). Relácia sa pomenováva názvom vzťahu, ktorý reprezentuje ¹ .
	Spojenie entít a relácie s kvantifikáciou	Ciselné ohodnotenie relácie (počtu výskytov entít v danej relácii) hodnotami z intervalu <min,max>, kde min je 0 alebo 1, max je 1 alebo viac (n).

V Yourdonovej notácii sa relácie ohodnocujú len hodnotou max pomocou 1 a n. Dolná hranica je reprezentovaná povinnosťou alebo voliteľnosťou relácie – povinná relácia je označená bodkou na konci hrany (viď. Obr. 4.6).

Ďalej nasleduje príklad entitno-relačného modelu systému - pracoviska modelovaného z hľadiska vzťahu pracovníkov a ich úloh.



• Obr. 4.3 Relačný model dát vyjadrený pomocou ERD v Chennovej notácii

V uvedenom dátovom modele sa modeluje vzťah medzi entitami: PRACOVNÍK, ZARADENIE, PRACOVISKO, KROK, ÚLOHA. z modelu vyplývajú nasledujúce skutočnosti (sémantika):

¹ Relácia nemusí byť symetrická, preto názov relácie je vhodné voliť pre všetky relácie tak, aby sa "čítili v jednom smere", napr. vždy proti smeru exportu kľúča (viď. Obr. 4.4, Obr. 4.5).

- na každej úlohe môže pracovať jeden alebo viac pracovníkov, úloha môže byť aj nepridelená
- každý pracovník môže pracovať na jednej alebo viacerých úlohách, alebo nemá priradenú žiadnu úlohu
- každý pracovník má práve jedno zaradenie, na jednom zaradení môže byť jeden alebo viac pracovníkov alebo dané zaradenie nemusí byť obsadené
- každá úloha sa môže, ale nemusí skladať z niekoľkých krokov, každý krok sa ale vždy viaže práve k jednej úlohe

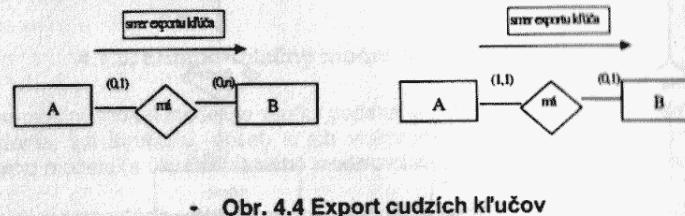
Metóda relačného modelovania sa používa v CASE systémoch strednej úrovne (middle CASE) – pre vytvorenie dátového modelu a pre modelovanie štruktúry dátových pamäti v procesnom modele dátových tokov.

V niektorých notáciách sa v diagrame ako uzly reprezentujú aj atribúty, pričom sa vizuálne rozlišujú neklúčové a klúčové atribúty (primárne klúče).

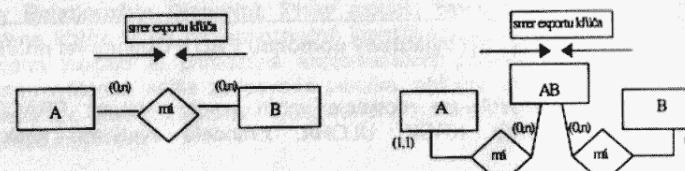
V ERD sa spravidla explicitne špecifikujú iba atribúty tvoriaci primárny klúč (jednoduchý alebo zložený). Cudzie klúče sú dôsledkom relačného vzťahu - ich miesto a význam je obyčajne možné odvodiť na základe relačného vzťahu medzi entitami. T.j. ich rozmiestnenie - export cudzích klúčov vyjadrujeme v ERD pomocou relácií. Cudzí klúč vznikne prenosom primárneho klúča z tzv. hlavnej - master entity do entity detail - tu sa tento atribút stane cudzím klúčom. Uvedený presun sa zvykne označovať ako export cudzích klúčov.

Smer exportu cudzeho klúča je pre reláciu $(0,1) : (0,n)$ od entity s násobnosťou výskytu $(0,1)$ k entite s násobnosťou výskytu $(0,n)$. Podobne je to pre reláciu $(1,1) : (0,n)$. T.j. **export cudzich klúčov je v smere od výskytu 0 alebo 1 k výskytu n-násobnému**.

Pre reláciu $(1,1) : (0,1)$ bude export cudzeho klúča od entity s výskytom práve 1 k entite s výskytom 0 alebo 1 krát. T.j. export cudzich klúčov v relácii bez jediného n-násobného výskytu je od entity s povinným výskytom (práve 1 krát) k entite s voliteľným výskytom (0 alebo 1 krát), vid. Obr. 4.4. Relácia medzi entitami A, B typu $(0,n) : (0,n)$ sa v fyzickom dátovom modele nahradí novou entitou AB a dvomi reláciami $(1,1) : (0,n)$ medzi entitami A a AB a $(0,n) : (1,1)$ medzi entitami AB a B. Smer exportu je potom definovaný z entity A do AB a z entity B do AB.



Obr. 4.4 Export cudzich klúčov



Obr. 4.5 Implementácia relácie m : n

Relácie $(0,1) : (0,n)$, $(1,1) : (0,n)$ medzi entitami A, B sa niekedy označujú aj ako **relácie typu 1 : n**. Príčom v 1. pripade bude k entite B existovať najviac 1 výskyt entity A (tzv. **nepovinná väzba**), v druhom pripade bude k entite B vždy existovať práve jedna entita A (tzv. **povinná väzba**). Relácie typu $(0,n) : (0,n)$, $(1,n) : (0,n)$ a podobne sú **relácie typu n : n** (alebo označované aj $m : n$). Na fyzickej úrovni je takto relácia medzi entitami A, B implementovaná pomocou dvoch relácií $1 : n$ medzi entitou A a pomocnou entitou AB a entitou B a pomocnou entitou AB (vid. Obr. 4.5). Z hľadiska modelovania vzťahov medzi entitami majú význam špeciálne relácie vyjadrujúce napr.:

- Závislosť existencie jednej entity od existencie druhej entity - tento vzťah sa zvykne označovať ako **slabá väzba (weak relation)**. Medzi entitami A a B je slabá väzba s reláciou $(1,1) : (0,n)$ práve teda, ak entita B (slabá entita) nemôže existovať bez príslušnej entity A (silná entita). Relácia sa implementuje tak, že cudzí klúč referujúci na silnú entitu bude súčasťou primárneho klúča v slabej entite.

Slabou reláciou môžeme napr. modelovať vzťah medzi entitou Objednávka a entitou Položka_objednávky. Položka objednávky nemôže existovať bez príslušnej objednávky a preto súčasťou klúča položky, môže byť číslo objednávky (primárny klúč entity Objednávka).

- Vzťah inkluzie - **supertyp a subtyp** - ak v každej z entít AB1, AB2, ..., ABn je rovnaká spoločná podmnožina atribútov A vrátane primárneho klúča, je možné túto spoločnú časť modelovať ako samostatnú entitu A so vzťahom k redukovaným entitám B1, B2, ..., Bn vytvorených z entít AB1, AB2, ..., ABn vynechaním spoločnej podmnožiny atribútov (okrem primárneho klúča).
- Rekurzívna relácia – relácia medzi výskytmi entít tohto istého typu.
- Výlučná relácia 1 z n – relácia danej entítie k práve jednej z n ďalších entít.

- ERD v notácii podľa Oracle*Method používa nasledujúce uzly a hrany:

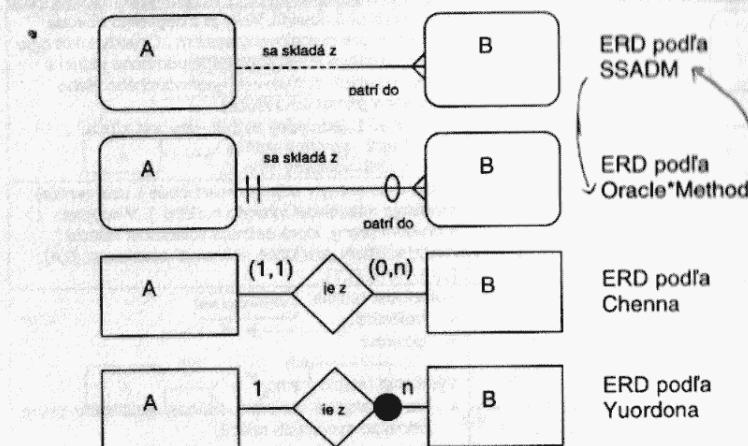
PRVOK	NÁZOV	MODELUJE
	Dátová entita	Typ údajového objektu, ktorý je z nejakého dôvodu významný pre modelovaný systém. Objekty tohto typu sú jednoznačne identifikovateľné pomocou jedného alebo viacerých atribútov (t.j. jednoduchého alebo zloženého primárneho klúča).
	Arita (násobnosť) relácie: n 1	# atr1 jedinečný atribút – súčasť klúča * atr2 povinný atribút o atr3 voliteľný atribút Ukončenie hran v pripojovacom bode k uzlu (entite) modeluje násobnosť výskytu n alebo 1. V spojení s druhom hran, ktorá definuje voliteľnosť relácie medzi entitami, je možné definovať násobnosť $(0,n)$, $(1,n)$, $(0,1)$, $(1,1)$.
	Relácia	Voliteľnosť relácie: • voliteľná • povinná Výlučnosť relácie 1 z n: • hrany spojené oblúkom definujú existenciu práve jednej zo spojených relácií.

Hrana spájajúca dve entity je rozdeľená na dve časti, z ktorých každá môže zodpovedať voliteľnej alebo povinnej relácii. Ak hrana spája entity A a B, potom časť hrany, ktorá vychádza z entity A, definuje voliteľnosť relácie entity A k entite B a naopak (viď. priklad na Obr. 4.6), každú časť je možné ohodnotiť názvom – relácia má pomenovania v oboch smeroch.

ERD v notácii podľa SSADM používa nasledujúce uzly a hrany:

PRVOK	NÁZOV	MODELUJE
	Dátová entita	Typ údajového objektu, ktorý je z nejakého dôvodu významný pre modelovaný systém. Objekty tohto typu sú jednoznačne identifikovateľné pomocou jedného alebo viacerých atribútov (t.j. jednoduchého alebo zloženého primárneho klúča).
	Arita (násobnosť) relácie: (0,n)	Ukončenie hrany v pripojovacom bode k uzlu (entite) modeluje násobnosť výskytu (súčasne aj voliteľnosť relácie): • nula alebo viackrát (voliteľná relácia)
	(1,n)	• jedna alebo viackrát (povinná relácia)
	(0,1)	• nula alebo jedenkrát (voliteľná relácia)
	(1,1)	• práve jedenkrát (povinná relácia)
	Relácia	Modeluje reláciu medzi entitami, ktoré spája. Relácia môže byť ohodnotená názvom v oboch smeroch.

Nasledujúci obrázok ilustruje základné rozdiely v jednotlivých notáciách ERD. Medzi entitami A a B je nasledujúci vzťah: ku každému výskytu entity B existuje práve jeden výskyt entity A a ku každému výskytu entity A existuje nula alebo viac výskytov entity B.



Obr. 4.6 Príklad ERD v rôznych notáciách

4.1.4. Objektový model

Paralelný vývoj dátových a procesných (funkčných) modelov systému môže viesť k divergencii dátového a funkčného modelu. Stáva sa to vtedy, ak nie je riadením projektu zabezpečená dostatočne častá konfrontácia a zosúladenie modelov v oblasti vzájomných súvislostí (viď. kap.5). Zosúladenie vyžaduje dodatočné úsile a čas, niekedy dlhší, ako bol čas ušetrený paralelným vývojom. Problém divergencie dátového a procesného modelu odstraňuje použitie objektového modelu systému (OM). V tomto modele sa stáva základným štrukturálnym prvkom spojenie dátovej entity so všetkými súvisiacimi funkciemi - zapúzdrenie funkcií a dát do objektu. Atribúty pôvodnej dátovej entity sa stávajú atribútmi objektu a definujú sledované vlastnosti objektu. Funkcie spojené s dátovou entitou definujú správanie objektu. Množina všetkých objektov s rovnakými druhmi sledovaných vlastností (rovnakými atribútmi) a rovnakým správaním (rovnakými funkciami) tvoria triedu. Objekt je inštanciou, konkrétnym výskytom danej triedy. Taktôľ definovaný pojem triedy je totožný s pojmom abstraktný údajový typ (AUT) – spojenie údajových štruktúr s definíciou všetkých operácií nad ňou. Triedy a objekty v objektovo-orientovanej analýze a návrhu (OOA/D) využívajú ale aj ďalšiu z vlastností objektov a tried (okrem zapúzdrenia) známe z paradigm objektovo-orientovaného programovania (OOP)² - je to dedičnosť.

Modelovanie systému pomocou tried a objektov prináša okrem odstránenia divergencie DM a FM aj ďalšie výhody:

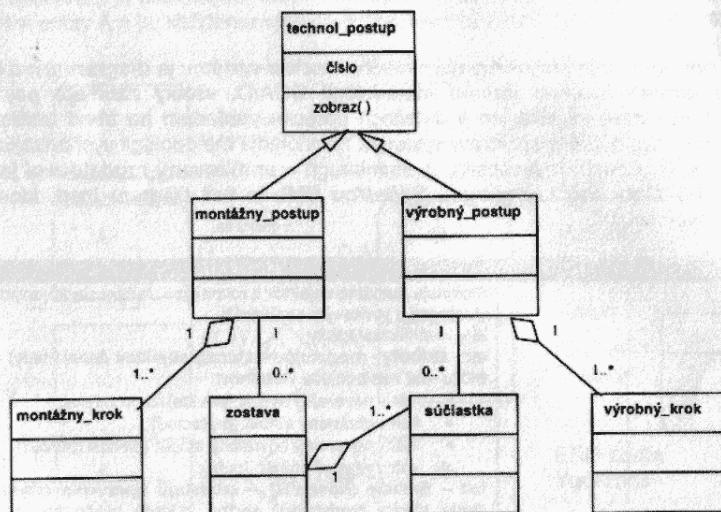
- pojem objektov je bližší realite (a tým aj používateľovi), ako analytické pojmy dátových entít a funkcií, je jednoduchšia verifikácia správnosti modelov – sú používateľovi zrozumiteľnejšie a lepšie sa k nim vedia vyjadriť
- od objektov v analytických modeloch je priamočiarý prechod k objektom v implementácii pomocou jazykov OOP (implementácia objektovo modelovaných systémov môže ale byť aj v jazykoch štruktúrovaného a modulárneho programovania).

Nástrojom pre vytvorenie statického objektového modelu systému je diagram tried (CD - Class Diagram). Existuje viacero metodológií OOA/D, všetky zahrňujú použitie diagramov tried, ktoré sa líšia len v detailoch notácie. Vzhľadom na trend unifikácie modelovacieho jazyka je perspektívny výsledok zjednotenia metodológií vychádzajúcich z prác autorov G. Booch, I. Jacobson, J. Rumbaugh – unifikovaný modelovací jazyk (UML- Unified Modeling Language). Súčasťou UML je tiež diagram tried, ktorého notácia je nasledovná:

PRVOK	NÁZOV	MODELUJE
	Trieda	<p>Modeluje množinu objektov s rovnakými druhmi sledovaných vlastností a rovnakým správaním.</p> <p>id - identifikátor triedy</p> <p>atr - atribúty - modelujú vlastnosti objektov danej triedy) – môžu mať nasledujúcu viditeľnosť:</p> <ul style="list-style-type: none"> -atr neverejný atribút (private) #atr chránený atribút (protected) /-atr neverejný odvodený atribút (private derived) +atr verejný atribút (public) <p>fun - funkcie (operácie) – modelujú správanie objektov danej triedy, predstavujú služby, o ktoré môže byť objekt požiadany. Môžu mať nasledujúcu viditeľnosť:</p>

² Vlastnosti OOP sú zapúzdrenie (encapsulation), dedičnosť (inheritance), polymorfizmus

		<ul style="list-style-type: none"> -fun neverejná funkcia (private) #fun chránená funkcia (protected) +fun verejná funkcia (public)
—	Asociácia	Vázba medzi triedami, ktorá modeluje komunikačný kanál medzi objektami danej triedy. Cez tento kanál je možná výmena informácií medzi objektmi, napr. posielaním správ.
rola	Rola	Rola - bližšie identifikuje existujúcu vásbu, mechanizmus alebo subjekt, ktorý ju sprostredkuje.
◇	Agregácia	Vázba medzi triedami, ktorá modeluje vzťah medzi celkom a časťou (koniec hrany zakončený kosoštvorcovom sa prípaja k triede predstavujúcej celok).
←	Dedenie	Vásba medzi triedami, ktorá modeluje dedičnosť medzi triedami. Umožňuje vytvoriť hierarchiu od všeobecných tried po triedy špecializované. Spojenie tried v smere trojuholníkovej šípky predstavuje zovšeobecnenie - generalizáciu a spojenie v opačnom smere predstavuje špecializáciu. Jedna trieda môže dediť vlastnosti a správanie aj z viacerých tried.
n — m	Násobnosť	Konec hrán sú ohodnotené číslom alebo intervalom modelujúcim počet objektov, ktoré môžu vystupovať v danej vásbe. Najčastejšie to je násobnosť: <ul style="list-style-type: none"> • práve jedna inštancia triedy • žiadna alebo jedna inštancia triedy • žiadna alebo viac inštancií triedy • jedna alebo viac inštancií triedy
→	Navigačnosť	Šípka modeluje jednosmernú vásbu (všetky vásby bez šípky sú obojsmerné).



Obr. 4.7 Príklad neúplného diagramu tried

4.2. Modelovanie dynamických vlastností programových systémov

Programové systémy majú dynamické vlastnosti už preto, že nejakým spôsobom komunikujú v čase so svojim okolím – dynamické vlastnosti súvisia s časom.

Napr. na dialógové informačné systémy existujú 3 základné všeobecné požiadavky zodpovedajúce ich 3 základným vlastnosťiam:

- poskytovať špecifikované aktuálne informácie
- poskytovať ich na základe danej požiadavky - udalosti
- poskytovať ich v požadovanom čase (od vzniku danej udalosti).

V riadiacich systémoch sú základné požiadavky a vlastnosti analogické, týkajú sa ale poskytovania potrebných riadiacich signálov.

Dynamické funkčné modely popisujú modelovaný systém z hľadiska jeho funkčných vlastností, medzi ktoré patrí najmä:

- tok dát a ich spracovanie - rozdelenie spracovania na procesy (funkcie), poskytované služby
- architektúra systému (dekompozícia na podsystémy a vásby medzi podsystémami), vymedzenie hraníc systému - definovanie okolia systému
- komunikácia systému s okolím - špecifikácia udalostí z okolia, na ktoré musí systém reagovať a spôsob tejto reakcie
- modulárna štruktúra systému a komunikácia medzi modulmi
- detailná štruktúra algoritmov

Medzi najbežnejšie metódy a nástroje funkčného modelovania patria:

- metóda modelovania dátových tokov (podľa De Marco, Gane&Sarson) – diagramy dátových tokov (DFD – Data Flow Diagrams), vid. kap. 4.2.1,
- metóda štruktúrovaného návrhu modulárnej štruktúry systému (podľa Yourdona a Constantinea) – diagramy modulárnej štruktúry (SCD – Structure Chart Diagrams), vid. kap. 4.2.2,
- metóda štruktúrovaného návrhu (podľa Jacksona) – (JSD – Jackson Structure Diagrams), vid. kap. 4.2.2.

Riadiace modely sú svojou podstatou dynamické - popisujú spôsob riadenia vykonávania systému alebo jeho časti z pohľadu používateľa alebo z pohľadu vnútornej štruktúry systému (vid. kap. 4.2.3). Medzi najčastejšie používané nástroje modelovania riadenia patria:

- stavové diagramy (STD - State Transition Diagrams)
- diagramy postupnosti obrazoviek (FSD - Forms Sequence Diagrams)
- diagramy scenárov (SD - Scenario Diagrams)

Dynamické dátové modely popisujú zmeny súvisiace s reprezentáciou dát, napr.:

- **Zmena stavu:** Hodnoty niektorých atribútov dátových entít a objektov ovplyvňujú ich vlastnosti natoľko, že hovoríme o zmene ich stavu – postupnosť týchto stavov tvorí životný cyklus. V prípade objektov modelujeme zmeny ich stavov pomocou stavového modelu (vid. kap. 4.2.3), pre dátové entity sa pre obdobný model používa označenie model životného cyklu entít (vid. kap. 4.2.4). Zmeny hodnôt atribútov sa dejú v rámci určitého definičného oboru – domény. Dynamické stráženie zachovania hodnôt atribútov v ramci ich domény definujú pravidlá doménovej integrity (vid. kap. 4.2.5).

- Zmena referencií:** Väzby medzi inštanciami dátových entít a tried (objektami) a samotná existencia entít a objektov sa počas využívania systému môže dynamicky meniť (objekty, entity vznikajú, zanikajú, vznikajú a zanikajú väzby medzi nimi). Dynamickú aktualizáciu referencií medzi dátovými entitami a objektami definujú pravidlá referenčnej integrity (vid. kap. 4.2.5).
- Zmena prístupovej doby k dátam:** v závislosti od štruktúry dátového modelu (vzájomných väzieb), formy utriedenia a od rozsahu dát (napr. od mohutnosti tabuľiek) sa môže významne meniť čas potrebný na sprístupnenie potrebných dát (čas od okamihu, keď sú zname požiadavky, po okamih, keď sú sprístupnené zodpovedajúce dátu. S prístupovou dobou potom súvisí aj doba potrebná na realizáciu požadovanej služby systému.

4.2.1. Model dátových tokov

Grafickým nástrojom pre model dátových tokov je **diagram dátových tokov (DFD - Data Flow Diagram)**. Existuje viacero notácií, ktoré sa v podstate líšia len vizuálnou reprezentáciou diagramov - tvarom symbolov, ich sémantika je rovnaká. Metóda podľa De Marca bola prvým použitím DFD pre analýzu informačných systémov (metóda bola uverejnená r. 1978). Iné symboly majú napr. DFD podľa DeMarco, Yourdon/DeMarco, Gane/Sarson atď. Niekoľko sa používa pre tento diagram aj názov - **bublinový diagram** (Bubble Chart) podľa represenácie procesov pomocou kruhových uzlov.

DFD je orientovaný sietový graf, jeho prvky tvoria:

- uzly:** externá entita (Terminator, External Entity, terminátor), proces (Process, Function, funkcia), elementárny proces, dátové pamäti (Datastore, DS, bázy dát, sklady dát)

- hrany:** údajový tok (Dataflow, DF)

Diagram dátových tokov je **hierarchicky dekomponovateľný** pomocou dekompozície procesov. Dekompozícia procesu je znova DFD na nižšej úrovni abstrakcie - modeluje daný proces s vyššou úrovňou detailizácie. Procesy, ktoré nie je potrebné ďalej dekomponovať, sa nazývajú **elementárne procesy** (elementárne funkcie). Elementárne procesy by mali odpovedať činnostiam, ktoré je možné jednoznačne popísať niekoľkými riadkami textu (minišpecifikácia), pre zostavenie tohto textu sa používa podmnožina prirodzeného jazyka alebo nejaký formálny jazyk (štruktúrovaná slovenčina, programovací jazyk, nejaký matematický formalizmus a pod.). Systém na najvyšej úrovni abstrakcie môže byť reprezentovaný jedným procesom. Pre zabezpečenie zrozumiteľnosti modelu sa na jednej úrovni dekompozície nepoužíva viac ako 7-9 procesov.

Pomocou funkčného (procesného) modelu je možné špecifikovať hranice systému. **Okolie systému** (kontext) tvoria externé entity, samotný systém tvoria všetky ostatné použité prvky modelu. Pre modelovanie väzieb systému ako celku na okolie sa preto využíva práve DFD na najvyššej úrovni dekompozície (0. úroveň) s jedným procesom reprezentujúcim celý systém. Tento diagram sa označuje ako **kontextový DFD**. Jeho dekompozíciu môžeme vyjadriť rozdelením systému na podsystémy - preto diagram na 1. úrovni dekompozície sa nazýva **systémový DFD**.

Reprezentácia DFD podľa De Marco a Gane-Sarson používa nasledujúce grafické prvky (ich sémantika je rovnaká aj v iných notáciách):

PRVOK podľa De Marco	PRVOK podľa Gane-Sarson	NÁZOV	MODELUJE
		Externá entita - terminátor (External entity, Terminator)	Subjekt alebo objekt, ktorý s modelovaným systémom komunikuje (napr. používateľ, iný systém, organizácia, riadiaci systém a pod.) Pritom nás nezaujíma štruktúra externej entity, ani jej vzťah k iným extermým entitám, ale len komunikácia s modelovaným systémom.
		Proces - funkcia (Process)	Činnosť, ktorá predstavuje spracovanie alebo nejakú manipuláciu s údajmi (vstup, výstup, prenos a pod.). Procesy je možné hierarchicky dekomponovať opäť do diagramu DFD. Procesy na najnižšej úrovni dekompozície sa nazývajú elementárne procesy. Každý proces musí mať aspoň jeden vstupný a aspoň jeden výstupný dátový tok.
		Dátová pamäť (Datastore)	Prostriedok, ktorý slúži na uchovávanie dát. Je možné modelovať pamäť dát v elektronickej podobe (databázy, súbory), ale aj papierové kartotéky. Každá dátová pamäť musí mať aspoň jeden vstupný a aspoň jeden výstupný dátový tok.
		Dátový tok (Dataflow)	<p>Prenos údajov je jedno alebo obojsmerný:</p> <ul style="list-style-type: none"> medzi externou entitou a procesom modeluje: vstup/výstup dát, udalosti, na ktoré musí systém reagovať, vonkajší prejav reakcie na tieto udalosti medzi dvoma procesmi modeluje: prenos dát medzi procesmi, vzájomnú závislosť procesov, prenos dočasných dát medzi procesom a dátovou pamäťou modeluje čítanie, zápis-ukladanie, ničenie-vymazávanie, aktualizáciu údajov v dátových pamätiach

Pôvodne bol diagram podľa De Marco určený nielen pre **modelovanie toku dát**, ale aj **modelovanie materiálového toku**. Procesy, terminátor, dátové pamäti, aj dátové toky sa označujú názvami a/alebo skratenými identifikátormi. Vzhľadom na to, čo jednotlivé prvky diagramu modelujú, je zaužívané nasledovné pomenovanie pomocou názovov a/alebo identifikátorov:

Prvok	Slovný druh pre identifikátor	Sémantika
externá entita	podstatné meno	názov, označenie subjektu alebo objektu
proces	slovesný tvar	vyjadrenie činnosti, postupu alebo služby, ktorú proces zabezpečuje (alebo vyjadrenie skupiny týchto činností, postupov)
dátová pamäť	podstatné meno	názov skupiny dát, dátovej entity, tabuľky, súboru, databázy, dátového servra, kartotéky,...
dátový tok	podstatné meno	názov skupiny dát, dátovej entity, dátového elementu, dokladu, tlačiva, obrazovkového formulára, vstupnej alebo výstupnej informácie,...

Okrem identifikátorov sa používa aj **číselné označenie prvkov**, hlavne procesov. Pretože je možná hierarchická dekompozícia DFD dekomponovaním procesov, používa sa jednoznačné číselné označenie procesov v tvare N1.N2.N3... vyjadrujúce ich hierarchiu resp. dekompozíciu nasledovne: proces s číslom 1.2.3 vznikol dekompozíciou procesu s číslom 1.2., procesy s číslami 1, 2, 3, ... vznikajú dekompozíciou procesu 0.

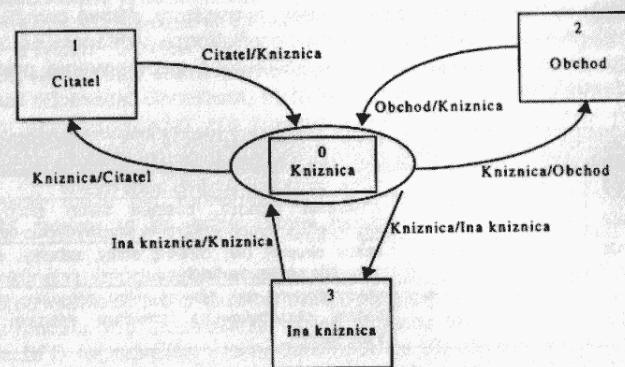
Procesný model neslúži pre popis časovej následnosť vykonávania jednotlivých činností (procesov, funkcií). Časovú následnosť ale určitým spôsobom je možné modelovať pomocou DFD (dátový tok spájajúci priamo dva procesy) alebo pomocou DFD rozšírených o riadiace procesy a riadiace toky. v DFD sa predpokladá, že proces (funkcia) sa vykoná, keď má k dispozícii všetky vstupy (hodnoty vstupných dátových tokov). z toho je zrejmé, že časovú následnosť je možné modelovať použitím dátových tokov medzi procesmi. Pri použití rozšírených DFD pre modelovanie riadenia sú riadiace toky generované riadiacim procesom. Logika tohto procesu môže byť modelovaná napr. stavovým diagramom. Vstup riadiaceho toku do procesu modeluje jeho odštartovanie.

V etape analýzy existujúceho a návrhu (syntézy) nového systému sa využíva DFD pre modelovanie procesov transformujúcich údaje:

- na fyzickej úrovni, napr. modelovanie existujúceho systému, modelovanie nového systému (rozdelenie spracovania na procesory, rozdelenie bázy dát v distribuovanom prostredí)
- na logickej úrovni - v etape syntézy nového systému bez zohľadnenia implementačných detailov.

DFD umožňuje uplatniť metódu štruktúrovania problémov (a celého systému) **zhoradol**. Model existujúceho systému sa vytvára pre jeho lepšie pochopenie, funkčný model nového systému sa vytvára pre definovanie spôsobu, akým systém bude riešiť používateľské požiadavky - definovanie interaktívnych a neinteraktívnych procesov, ich vstupov a výstupov. Okrem dekompozície procesov sa používa aj dekompozícia dátových tokov a dátových pamäti do vhodného dátového modelu. U dátových tokov obyčajne potrebujeme modelovať ich štruktúru (hierarchický model dát reprezentovaný napr. pomocou stromového diagramu dátových štruktúr), u dátových pamäti modelujeme ich štruktúru a relačné vzťahy medzi dátovými entitami (relačný model dát). Metóda modelovania dátových tokov sa používa v CASE systémoch strednej úrovne (middle CASE) - pre podporu analýzy/návrhu systémov.

Príklad neúplného logického procesného modelu informačného systému knižnice (kontextový a systémový DFD bez ďalšej dekompozície):



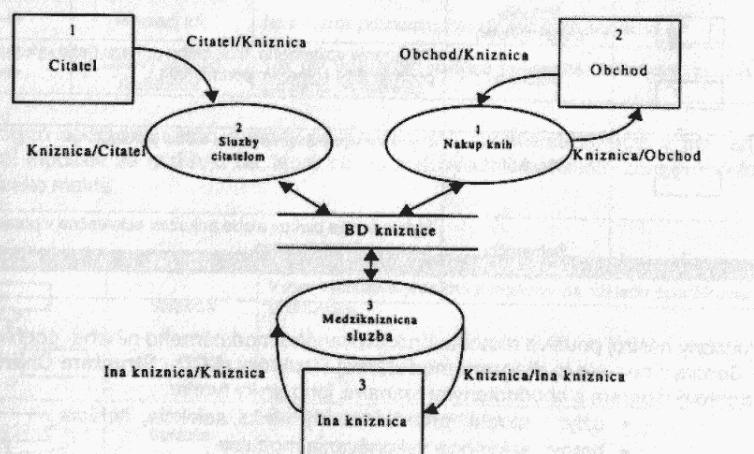
Obr. 4.8 Kontextový DFD pre systém Knižnica

Uvedený kontextový diagram modeluje vzťahy systému (knižnice) k okolia - v uvedenom (pre tento príklad zjednodušenom) modele toto okolie tvoria iné knižnice, čitatelia knižnice a obchodné organizácie. Toky predstavujúce komunikáciu medzi okolím a systémom sú identifikované zjednodušene v tvaru Zdroj/Ciel. Obyčajne zahrnuje tento tok veľké množstvo rôznych druhov dátových štruktúr a elementov, ich dekompozícia môže byť popísaná v rámci dekompozície DFD pomocou detailnejších tokov a/alebo pomocou dátového modelu daného toku.

Dátové toky, ktoré do systému vstupujú, je možné interpretovať ako **udalosti**, na ktoré musí systém reagovať. Touto reakciou môže byť generovanie nejakého výstupného toku a/alebo zmena vnútorného stavu systému (napr. reprezentovaná zmenou uchovávaných dát). Kontext systému je možné modelovať aj pomocou **maticy udalostí** (ELM, event list matrix) s riadkami popisujúcimi udalosti (vstupné toky z okolia do systému) s nasledujúcimi informáciami (stĺpcami):

- meno udalosti (názov vstupného toku)
- zdroj udalosti (externá entita)
- typ udalosti (napr. dočasná - nevidovaná systémom, len spôsobí príslušnú reakciu, databázová - okrem reakcie systému je vždy aj presne evidovaná a pod.)
- reakcia systému na udalosť (napr. výstup dát, zmeny v báze dát a pod.)

Nasledujúci diagram dátových tokov je dekompozíciou procesu predstavujúceho celý informačný systém knižnice. Je to tzv. systémový diagram, ktorý v našom prípade modeluje jednotlivé skupiny poskytovaných služieb.



Obr. 4.9 Systémový DFD

Určitou formou diagramov dátových tokov sú tzv. diagramy činností (UCD - Use-Case Diagram³) používané v objektových metodológiach. Prvkami UCD sú uzly: používateľia

³ Use Case Model Ivara Jacobsona [13] modeluje funkcia systému z pohľadu používateľa.

systému (actor, v DFD terminátor) a činnosti (use case, v DFD procesy), hrany: väzby medzi používateľom a činnosťou (dátové toky).

4.2.2. Model štruktúry algoritmu

Grafickým nástrojom pre modelovanie štruktúry algoritmov je **diagram štruktúry algoritmu** (Jackson Structure Diagram⁴, JSD). JSD je stromový diagram, jeho prvky tvoria:

- uzly: blok (sekvencia jednoduchý príkaz alebo blok príkazov), voliteľný príkaz alebo blok (vyjadrenie selekcie), opakovane vykonávaný príkaz alebo blok (iterácia)
- hrany: dekompozícia štruktúry algoritmu (na sekvencie, selekcie, iterácie)

Koreňom stromu je uzol zodpovedajúci programu, modulu alebo podprogramu, ktorého štruktúra sa popisuje. Diagram štruktúry modeluje detailne použitie základných riadiacich štruktúr (sekvencie, selekcie, iterácie) v podprogramoch, moduloch a programe. Reprezentácia JSD používa nasledujúce grafické prvky:

PRVOK	NÁZOV	MODELUJE
	Príkaz Blok	Vykonanie definovanej činnosti alebo viacerých činností, (operácií) nad definovanými údajmi.
	Volanie podprogramu	Vykonanie (vyvolanie) podprogramu.
	Selekcia	Vykonanie voliteľného bloku alebo príkazu (jeho vykonanie je podmienené splnením podmienky).
	Iterácia	Cyklické vykonávanie bloku alebo príkazu.
	Sekvencia	Vykonávanie blokov alebo príkazov sekvenčne v poradí zhora-nadol a zľava-doprava.

Podobný nástroj používa metóda štruktúrovaného modulárneho návrhu podľa Yourdona a Constantinea - je to **diagram modulárnej štruktúry (SCD - Structure Chart)**. SCD je stromový diagram s ohodnotenými hranami, jeho prvky tvoria:

- uzly: modul, preddefinovaný modul, selekcia, iterácia
- hrany: sekvencia vykonávania modulov
- ohodnotenie hrán: dátový tok, riadiaci tok

Koreňom stromu SCD je uzol časti systému, ktorého štruktúra sa popisuje. Diagram modulárnej štruktúry modeluje detailne modulárnu štruktúru systému. Nelistové

⁴ JSD je súčasťou metódy JSP (Jackson Structure Programming) - je to metóda pre dátovo orientovaný detailný návrh štruktúry algoritmov (programov, podprogramov, modulov). Metóda vychádza z poznania úzkej závislosti medzi štruktúrou dát a použitými riadiacimi štruktúrami v algoritnoch.

(nokoncové) uzly predstavujú vyššiu abstrakciu činnosti, listové uzly zodpovedajú konkrétnym modulom zabezpečujúcim požadovanú činnosť. Hrany sú ohodnotené (renášanými alebo ináč sprístupňovanými) informáciami.

Reprezentácia SCD podľa Yourdona a Constantinea používa nasledujúce grafické prvky:

PRVOK	NÁZOV	MODELUJE
	Modul	Definovaná činnosť alebo skupina činností, (operácií) nad definovanými údajmi. Operácie môžu byť definované nejakou podmožinou programovacieho alebo iného formálneho jazyka.
	Preddefinovaný modul	Vyvolanie alebo vykonanie opakovane použiteľného modulu alebo podprogramu.
	Selekcia	Výber vykonania jedného z nasledujúcich modulov na základe špecifikovanej podmienky, stavu alebo udalosti.
	Iterácia	Cyklické vykonávanie nasledujúceho modulu (modulov).
	Sekvencia	Vykonávanie modulov alebo preddefinovaných modulov sekvenčne v poradí zhora-nadol a zľava-doprava.
	Dátový tok	Prenos parametrov, použitie globálnych premenných, prenos návratových hodnôt.
	Riadiaci tok	Nastavenie príznakov, prenos riadiacich signálov.
	Dátový a/alebo riadiaci tok	Tok dát (operandy, riadiace premenné, parametre, používané globálne premenné).

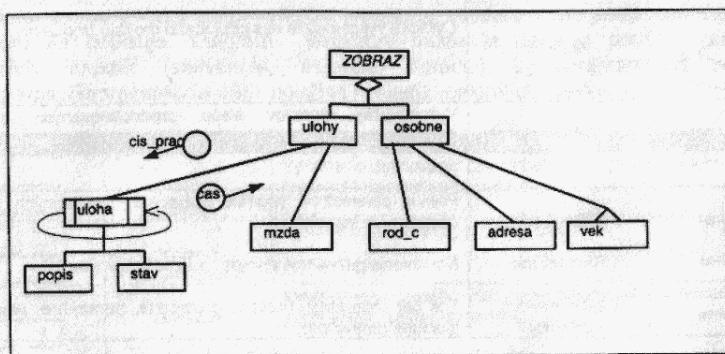
V niektorých aplikáciach tejto metódy sa používa iné označenie iterácie a sekvencie, a v rámci modulov sa rozlišujú tie, ktoré sa na zodpovedajúce miesto zdrojového textu vkladajú ako makrá:

PRVOK	NÁZOV	MODELUJE
	Selekcia	Výber vykonania jedného z modulov na základe špecifikovanej podmienky.
	Iterácia	Cyklické vykonávanie modulu.
	Vkladaný modul	Činnosť, ktorá nebude realizovaná volaním podprogramu, ale priamym vykonaním vloženej postupnosti príkazov.

Pre pomenovanie modulov sa používa identifikátor modulu, uzly iterácie sa označujú špecifikáciou cyklu (podmienka ukončenia cyklu, inicializácia cyku, príprava ďalšieho kroku), uzly selekcie sa označujú testovanou podmienkou, udalosťou, ktorá spôsobuje vvetvenie a pod.).

Diagramy SCD je možné vytvárať zhora-nadol s využitím dekompozície modulov - modul môže byť dekomponovaný opäť do diagramu SCD s koreňovým uzlom identifikovaným rovnako ako dekomponovaný modul.

Nasledujúci príklad modeluje štruktúru modulu **ZOBRAZ**, ktorý poskytuje dve služby: výpis zo zoznamu úloh (modul **ulohy**) pre vybratého pracovníka a výpis osobných údajov (modul **osobne**). Samotný výpis úloh realizuje podprogram **uloha**, ktorý na základe vstupného parametra **cis_prac** (identifikačné číslo pracovníka) v cykle zobrazí jednotlivé pridelené úlohy, ich popis a spočítá sumárny čas riešenia (výstupná hodnota **cas**). Táto hodnota sa ďalej použije v nasledujúcom výpočte mzdy. Modul **osobne** tvorí sekvencia blokov pre výpis rodného čísla adresy a veku vybratého pracovníka.



Obr. 4.10 Štruktúra modulu definovaná pomocou SCD

Metóda modelovania modulárnej štruktúry sa používa v CASE systémoch nižejúcej úrovni (lower CASE) - pre podporu detailného návrhu a implementácie programových systémov.

4.2.3. Riadiace modely

Softvérové systémy charakterizuje nie len tok dát, ale aj tok riadenia - poradie vykonávania jednotlivých činností (funkcií, procesov). Tokom riadenia je dané aj poradie spracovania dát, ktoré sú vstupom týchto procesov a poradie generovania výstupných dátových tokov. v modeloch popisujúcich štruktúru algoritmov (kap. 4.2.2) je tok riadenia definovaný pomocou sekvencie, selekcie, iterácie a procedurálneho volania. V modele dátových tokov (kap. 4.2.1) sa implicitne predpokladá, že daný proces sa vykoná, keď má k dispozícii všetky požadované vstupné toky. Tento spôsob spracovania nemusí presne zodpovedať požiadavkám kladeným na cieľový systém. Napr. informačný systém má poskytnúť požadované informácie po výbere tejto služby z ponuky a následnom vyplnení potrebných vstupných údajov. Výsledný efekt je taky istý, ako implicitne predpokladá model dátových tokov - vykonávanie služby (vlastné

spracovanie dát) sa v podstate odštartuje až po zadaní potrebných vstupov. z hľadiska riadenia je ale podstatné to, že proces zodpovedajúci danej službe sa štartuje na základe udalosti "výber služby z ponuky". v riadiacom systéme modelovanom pomocou DFD môže existovať požiadavka na odštartovanie nejakého procesu po splnení určitých podmienok, v určitom časovom okamžiku, t.j. opäť na základe nejakej udalosti. Požiadavka riadenia sa v rozšírených DFD rieši doplnením riadiacich procesov a riadiacich tokov. Objektívny model zostavený pomocou diagramu tried (CD) (kap. 4.1.4) obsahuje sice špecifikáciu komunikačných kanálov medzi objektami, nedefinuje ale spôsob riadenia tejto komunikácie.

Základom riadiacich modelov pre rozšírené DFD, aj modelov pre popis správania objektov sú konečnostavové automaty (KSA) a ich grafická reprezentácia – stavové diagraamy – používa sa aj pojem stavové modely. Alternatívnu formou stavových diagramov sú diagraamy scenárov a stromové diagraamy postupnosti obrazoviek.

Stavové modely popisujú konečný počet stavov systému alebo jeho časti a riadenie prechodov medzi stavmi pomocou stavových diagramov (STD - State Transition Diagram).

V diagramoch dátových tokov rozšírených o riadiace procesy a riadiace toky sa pomocou STD definuje spôsob generovania riadiacich signálov, t.j. činnosť riadiacich procesov - STD tvorí dekompozíciu riadiaceho procesu. (V rozšírenom DFD sú riadiace procesy a riadiace toky odlišené vizuálne napr. čiarkovanou čiarou).

Pri objektovom modelovaní sa STD vytvára pre všetky triedy, pre ktoré je potrebné modelovať správanie jej objektov.

STD je sieťový diagram s ohodnotenými hranami a uzlami. Môže zodpovedať konečnostavovým automatom typu Mealy alebo typu Moore, podľa toho, ktorý typ vystihuje správanie modelovaného systému alebo jeho časti. Pri oboch typoch sa predpokladá prechod medzi stavmi buď na základe rozpoznania nejakej vonkajšej vstupnej udalosti alebo na základe vnútormej aktivity (napr. vyhodnotenia nejakých podmienok). Pri modeloch na báze KSA typu Moore sa všetka činnosť vykonáva v rámci stavov, t.j. aj generovanie nových (výstupných) udalostí. Pri modeloch na báze KSA typu Mealy sa činnosť môže vykonávať aj pri realizácii prechodu – t.j. generovanie výstupných udalostí je pri vykonávaní prechodu.

Prvky STD tvoria:

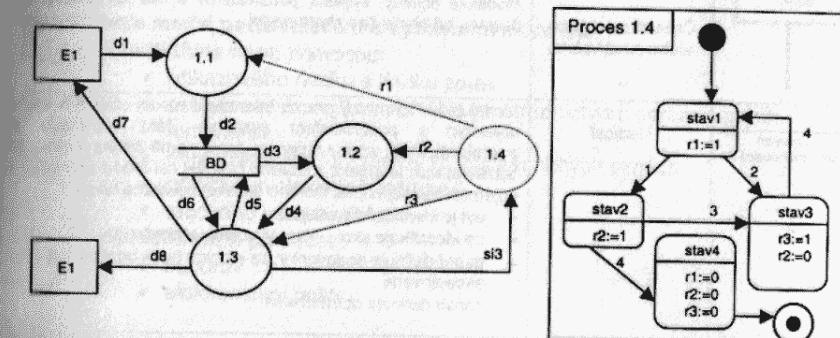
- uzly - stavы
- hrany - prechody medzi stavmi
- ohodnotenie hrán - udalosti spôsobujúce prechod a nové udalosti, prechodom vyvolané

Reprezentácia STD podľa Rumbaugh/OMT používa nasledujúce grafické prvky:

PRVOK	NÁZOV	MODELUJE
	Stav	Modeluje obdobie existencie objektu - stav, počas ktorého objekt splňa definované podmienky a vykonáva definovanú činnosť a/alebo čaká, až nastane definovaná udalosť. Činnosť alebo čakanie na udalosť tvoria aktívitu spojený s daným stavom - táto aktívita je definovaná pomocou zložky operácia.
	Trieda	Modeluje triedu objektov.
	Prechod (hrana medzi 2 stavmi)	<p>Modeluje prechod medzi stavmi. Ak je pomenovaný (evt), prechod sa vykoná na základe vonkajšej (vstupnej) udalosti, ináč sa prechod vykoná automaticky po ukončení aktívity stavu, z ktorého hranu vychádza.</p> <ul style="list-style-type: none"> • evt je identifikácia udalosti • op identifikuje akciu, ktorú vyvolala udalosť evt • guard definuje podmienky, za ktorých bude udalosť evt akceptovaná • constr definuje obmedzenia
	Správa	Modeluje odoslanie správy – mes s parametrami par triedy (objektom danej triedy). Ak správa spôsobí návrat nejakej hodnoty, modeluje ju ret s typom typ. Iniciátorom správy môže byť: <ul style="list-style-type: none"> • prechod medzi stavmi – odoslanie správy (mes1) modeluje orientovanú hranu spájajúcu hranu prechodu a uzol triedy • stav (správa mes2 sa odošle v rámci vykonávania činnosti v stave) – odoslanie takéjto správy modeluje hranu spájajúcu stav s triedou.
	Počiatkový stav (iniciátor)	Modeluje stav, ktorý je pre daný objekt iniciálnym stavom.
	Koncový stav (terminátor)	Modeluje koncový stav objektu.
	Hniezdenie stavov	Modeluje hierarchickú dekompozíciu zložitých stavov (nadstavov) na podstavy.
	História	Modeluje požiadavku začať vykonávanie v tom podstave, ktorý bol aktivovaný ako posledný pri predchádzajúcej aktivácii daného nadstavu označeného ikonou "história".

Jednoduchšia modifikácia STD na báze KSA typu Moore postačuje pre modelovanie generovania riadiacich signálov (riadiacich tokov) riadiacim procesom v rozširovnom DFD. Ak generované riadiace signály závisia len od aktuálneho stavu riadiaceho procesu a prechody medzi stavmi len od aktuálneho stavu a aktuálnych hodnôt vstupných dátových tokov tohto procesu (stavovo-informačné signály z riadeného procesu), potom hrany v DFD stačí ohodnotiť napr. hodnotami stavovo-informačných signálov a operáciu v stavoch bude generovanie riadiacich signálov.

V príklade na Obr. 4.11 je aktivácia procesov 1.1, 1.2, 1.3 modelovaná stavovým diagramom, ktorý je dekompozíciou (resp. detailným popisom) riadiaceho procesu 1.4. Stavovo-informačný signál si3 nadobúda celočíselnú hodnotu, riadiace signály sú v príklade dvojhodnotové (0, 1) – hodnota 1 spôsobí aktiváciu procesu, hodnota 0 deaktiváciu.



Obr. 4.11 Priklad pouzitia stavoveho diagramu s DFD

Pri objektovom modelovaní systému sa jeho činnosti reprezentujú pomocou interakcie objektov a používateľov prostredníctvom posielania správ. Diagram scenárov (SD – Scenario Diagram) modeluje systém z pohľadu časovej následnosti v interakcií medzi objektami systému navzájom a okolím systému (používateľmi).

Prvky diagramu scenárov tvoria:

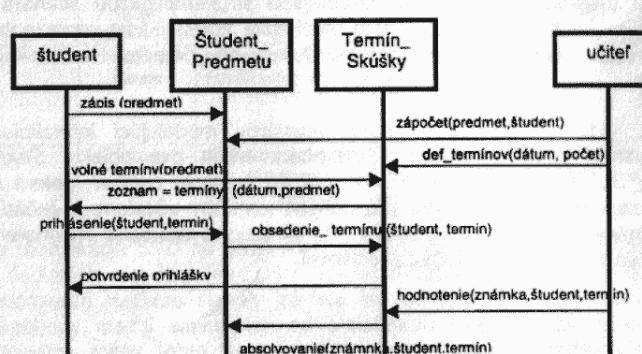
- paralelné časové osi objektov systému a používateľov
- orientované hrany spájajúce jednotlivé osi vo význame posielaných správ

Casovanie odosielania správ je dané pozíciou na časovej osi a relatívnu pozíciou voči polohe iných správ. Odosielateľ a adresát správy je daný počiatocným a koncovým bodom hrany. Pre modelovanie systému ako celku (iba z pohľadu používateľa) sa použije iba jedna časová os patriaca systému. Jej detailizáciou (dekompozíciou) môže byť ďalší diagram scenárov s podrobnejšou štrukturalizáciou scenára pre objekty. Podobne môže byť dekomponovaná zložitejšia komunikácia reprezentovaná jednou orientovanou hranou (abstraktnou správou) na detailnú komunikáciu v ďalšom diagrame.

Na Obr. 4.12 je príklad diagramu scenárov modelujúci interakciu podsystému s používateľmi učiteľ a študent. Pod systém tvoria dva objekty Študent_Predmetu a Termín_Skúšky. Študent sa musí najprv na predmet zapísť, potom mu môže byť učiteľ zápočet. Po definovaní skúšobných terminov učiteľom, si môže študent zistiť výhľad termínov a prihlásiť sa na skúšku. Po jej absolvovaní bude ohodnotený a v prípade úspechu sa zaevíduje, že skúšku absolvoval.

Reprezentácia SD podľa Rumbaugh/OMT používa nasledujúce grafické prvky:

PRVOK	NÁZOV	MODELUJE
	Časová os objektu alebo používateľa	Modeluje objekty, systém, používateľov a čas ich aktivity. Na časovej osi plynne čas zhora-nadol.
	Udalosť	Modeluje jednosmerný prenos informácie medzi objektami alebo objektom a používateľom systému. Táto informácia je o udalosti, ktorá vznikla v danom čase, nemá žiadne trvanie, jej odosielačom je objekt, z ktorého časovej osi hrana vychádza a príjemcom je objekt, na ktorého časovej osi hrana končí. <ul style="list-style-type: none"> • evt je identifikácia udalosti • op identifikuje akciu, ktorú vyvolala udalosť evt • guard definuje podmienky, za ktorých bude udalosť evt akceptovaná • constr definuje obmedzenia
	Správa	Modeluje odoslanie správy – mes s parametrami par triede (objektom danej triedy). Ak správa spôsobí návrat nejakej hodnoty, modeluje ju ret s typom typ. <p>Správa môže byť:</p> <ul style="list-style-type: none"> • abstraktná - predstavuje ďalší diagram scenárov • jednoduchá - predstavuje jednosmerné riadenie • synchronná - odosielač čaká, pokiaľ príjemca ju akceptuje • asynchronná - odosielač nečaká na odpoveď • časovo obmedzená - odosielač čaká na odpoveď len definovaný čas, ak potvrdenie, nepríde, správa sa ruší • synchronná bez čakania - ak nepríde odpoveď hneď po odoslaní, správa sa ruší
	Počiatocný stav (iniciátor)	Modeluje stav, ktorým sa interakcia začína.
	Koncový stav (terminátor)	Modeluje koncový stav, ktorým sa interakcia končí.



Obr. 4.12 Príklad diagramu scenárov

Interakciu programového systému s okolím, podľa potreby aj komunikáciu medzi podsystémami popisuje tzv. **komunikačný model**. Pretože okolie systému môže byť tvorené rozličnými subjektami a objektami, môže byť aj spôsob interakcie veľmi rôznorodý.

Komunikácia medzi používateľom a systémom môže byť:

- **interaktívna** napr. pomocou:
 - príkazového riadku a riadku správ
 - celoobrazovkovej komunikácie prostredníctvom obrazovkových formulárov
 - obrazovkových formulárov vo viacnásobných oknach
 - grafického oknového rozhrania
 - rečového vstupu/výstupu
- **dávková** napr. pomocou:
 - súborov
 - elektronickej pošty

Model komunikácie medzi používateľom a programovým systémom sa nazýva **model používateľského rozhrania (UIM - User Interface Model)**, resp. model dialógu medzi používateľom a systémom. UIM sa v súčasnosti najčastejšie zostavuje ako model ponukového systému (systém menu, následnosti obrazoviek - **diagram postupnosti obrazoviek (FSD - Forms Sequence Diagram)**). Je to acyklický orientovaný graf (zbortený strom). Jeho prvky majú nasledujúci význam:

uzly: **obrazovkové formuláre** (obrazovky, okná) alebo aktivácia poddialógu, koreňový uzol identifikuje daný dialóg alebo poddialóg
hrany: možné **prechody** medzi jednotlivými obrazovkovými formulárami (bývajú označené alternatívou z ponuky, pri výbere ktorej sa prechod vykoná, resp. udalosťou alebo podmienkou, pri splnení ktorej sa prechod vykoná).

Súčasťou modelu býva aj popis **formátu obrazoviek** s umiestnením a typom udajových a riadiacich polí, ich implicitným nastavením a pod. Pre zostavenie FSD sa používajú prvky s nasledujúcim tvarom a významom:

PRVOK	NÁZOV	MODELUJE
	Obrazovkový formulár (Form)	Celoobrazovkový formulár (obrazovka) alebo okno. Môže byť označený napr. názvom formulára, názvom modulu, služby, skupiny služieb a pod.
	Poddialóg	Volanie poddialógu.
	Prechod	Prechod medzi stavmi dialógu, formulárimi, oknami a pod. Hrana je ohodnotená podmienkou prechodu (udalosť, podmienka alebo vybraná alternatíva z ponuky, pri ktorej dochádza k prechodu k nasledujúcej obrazovke, oknu a pod.).

FSD súvisí s dialógovými procesmi a môže vytvárať dekompozíciu týchto procesov v diagrame dátových tokov. Pre modelovanie interakcie sa používa okrem stromového FSD aj sieťový diagram STD (korešpondencia medzi stromom a sieťovým diagramom je jednoznačná – strom reprezentuje kostru sieťového diagramu.)

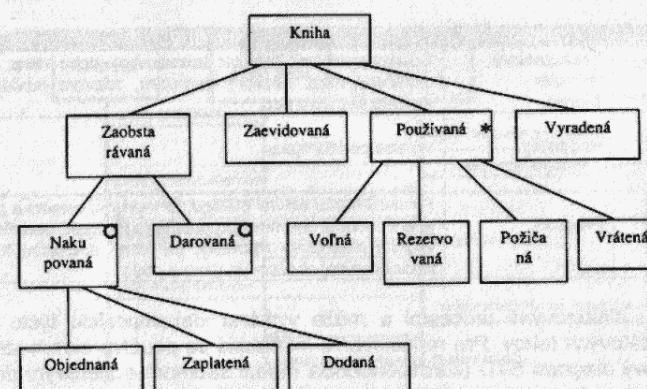
Komunikácia medzi dvoma programovými systémami môže byť realizovaná napr.:

- prístupom do spoločnej dátovej oblasti v operačnej pamäti
- prístupom do spoločnej dátovej oblasti na vonkajšej pamäti (bežný súbor súborového systému, vybraté tabuľky bázy dát - zdieľané alebo replikované tabuľky a pod.)
- prenosom správ (súborov s dohodnutou syntaxou, sémantikou a spôsobom synchronizácie)

Pre modelovanie takejto komunikácie sa dá využiť diagram dátových tokov, procesy môžu modelovať systémy a podsystémy. Pre modelovanie medzisystémovej komunikácie je možné použiť riadiace procesy s dekompozíciou pomocou stavových diagramov (viď. kap. 4.2.3) alebo diagramy scenárov.

o 4.2.4. Model životného cyklu entít

Pre modelovanie dynamických vlastností dátových entít sa používa **model životného cyklu entít (ELH - Entity Life History)**. Tento model popisuje stavy, ktorými entita prechádza od svojho vzniku - vstupu do systému, až po zánik - vyradenie zo systému. Pre grafickú reprezentáciu tohto modelu sa používa viacero metód, napr. Jacksonove diagrame (kap. 4.1.2), Petriho siete, stavové diagrame (kap. 4.2.3), PERT diagrame (kap. 6.7.3.2). ELH vo forme stromového Jacksonovho diagramu predstavuje životný cyklus v smere od koreňa zhora-nadol a zľava-doprava. Nasledujúci príklad ilustruje použitie Jacksonovho diagramu pre modelovanie životného cyklu entity *kniha* v informačnom systéme knižnice. Životný cyklus každej knižnej jednotky v uvedenom systéme začína zaobstaraním nákupom alebo darom a končí vyradením. Uzel 1 v diagrame je názov dátovej entity, ktorej životný cyklus modelujeme. Ostatné uzly predstavujú zľava-doprava stavov, ktorými táto entita prechádza. Stav 4 sa môže cyklicky opakovať v rámci tohto stavu entita prechádza povinnými podstavmi *volná, požičaná, vrátená* a nepovinným stavom *rezervovaná*.



Obr. 4.13 Príklad modelu životného cyklu entity "kniha"

4.2.5. Modelovanie zabezpečenia integrity bázy dát

Pre zachovanie **konzistencie bázy dát** počas jej používania pomocou služieb programového systému (napr. informačného systému) je potrebné definovať a implementovať pravidlá pre zabezpečenie doménovej a referenčnej integrity.

Doménová integrita (DI) je taká vlastnosť systému, ktorá zaručuje, že počas používania systému budú nadobúdať všetky neklúčové atribúty dátových entít v báze dát (resp. objektov) len prípustné hodnoty. T. j. doménová integrita sa týka **priprustnosti hodnôt neklúčových atribútov**. Táto integrita sa zabezpečuje implementáciou pravidiel doménovej integrity.

Pre definovanie pravidiel doménovej integrity sa používa pri popise atribútov entít (atlpov tabuľiek) okrem fyzického typu aj podmienka, ktorú hodnota musí splňať alebo sú uvedené všetky prípustné hodnoty atribútov alebo príznak toho, že atribút môže byť **ne**definovaný (príznak môže byť napr. konštantu *null*). Ak musí byť atribút vždy definovaný, je to indikované príznakom *not null* alebo uvedením prípustných hodnôt.

Referenčná integrita (RI) je taká vlastnosť systému, ktorá zaručuje, že pri používaní systému si bude báza dát zachovávať konzistenciu vnútorných väzieb v dátach - prístupnosť všetkých dát, ktoré majú v dátovom modele definované vzájomné väzby. T. j. v implementácii pomocou relačnej databázy budú cudzie klúče vždy referovať iba na existujúce výskyty dátových entít (resp. objektov) alebo nebudú referovať nikam (budú mať hodnotu null), ak je to prípustné. RI sa týka klúčových atribútov (primárnych a cudzích klúčov).

Referenčná integrita sa dá zabezpečiť implementáciou systému pravidiel referenčnej integrity – napr. pravidiel pre vykonávanie operácií nad relačnou implementáciou bázy dát: **vloženie riadku (I - insert), aktuálizácia riadku (U - update), zrušenie riadku (D - delete)** v tabuľkách bázy dát.

Pre definovanie pravidiel referenčnej integrity navrhovanej bázy dát je v popise klúčových atribútov entít uvedený požadovaný spôsob zabezpečenia referenčnej integrity pri operáciach I (insert), U (update), D (delete).

Akciaľné spôsoby - pravidlá zabezpečenia RI pre operácie I, U, D sú nasledujúce (bližší popis je v Tab. 4.1):

- **R - reštrikčný**: príslušná operácia je zakázaná, ak by malo dôjsť k narušeniu integrity
- **N - nulitný**: príslušná operácia je povolená, integrita sa zabezpečí nastavením hodnoty referujúcej "nikam" (*null*) v miestach, kde ju výsledok operácie narušil
- **K - kaskádny**: príslušná operácia je povolená, ale pre zabezpečenie integrity sa vykoná viacnásobne (kaskádne).

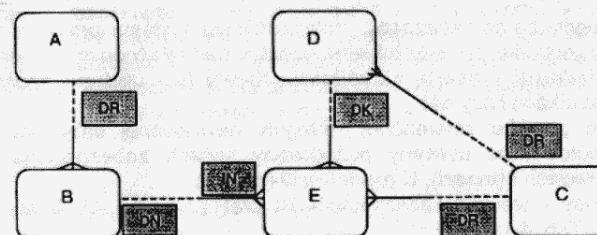
Pretiaž RI sa týka hodnôt primárnych (PK) a im zodpovedajúcich cudzích klúčov (FK), pravidlá sa definujú zvlášť pre primárny a cudzí klúč v každej dvojici entít (objektov) A, ktoré sú spojené reláciou. Ak smer exportu PK je v smere od A k B (v entite B je cudzí klúč referujúci na A), entitu A budeme nazývať rodičovskou (parent alebo hlavnou master), entita B bude dcérská (alebo vedľajšia - detail).

Pre jednotlivé operácie I, U, D v master (A) alebo detail (B) entite bude interpretácia pravidiel RI nasledovná:

ENTITA	OPERÁCIA	PRAVIDLO RI	OZNAČENIE	VÝZNAM PRI RELAČNEJ IMPLEMENTACII BÁZY DÁT
master	D	R	DR	Riadok v A tabuľke nemôže byť zrušený, pokiaľ existuje aspoň 1 riadok v B tabuľke, ktorý na neho referuje.
master	D	N	DN	Riadok v A tabuľke môže byť zrušený, ale s jeho zrušením sa v tabuľke B vo všetkých riadkoch nastaví príslušný cudzí klúč na hodnotu null.
master	D	K	DK	Riadok v A tabuľke môže byť zrušený, ale s jeho zrušením sa zruší aj všetky tie riadky v tabuľke B, ktoré referovali na zrušený riadok v A.
master	U	R	UR	Primárny klúč v riadku tabuľky A nemôže byť zmenený, ak existuje aspoň jeden riadok v B tabuľke, ktorý naňho referuje.
master	U	N	UN	Primárny klúč v A tabuľke môže byť zmenený, ale so zmenou sa vo všetkých riadkoch B tabuľky (ktoré referovali na riadok so zmeneným klúčom) nastaví zodpovedajúci cudzí klúč na novú hodnotu null.
master	U	K	UK	Primárny klúč v A tabuľke môže byť zmenený, ale so zmenou sa vo všetkých riadkoch B tabuľky (ktoré referovali na riadok so zmeneným klúčom) nastaví zodpovedajúci cudzí klúč na novú hodnotu null.
detail	I	R	IR	Do B tabuľky nie je možné zaradiť riadok s hodnotou príslušného cudzieho klúča takou, ktorá ešte neexistuje v tabuľke A ako primárny klúč.
detail	I	N	IN	Do B tabuľky je možné zaradiť riadok s hodnotou príslušného cudzieho klúča null.
detail	I	K	IK	Do B tabuľky je možné zaradiť riadok s hodnotou príslušného cudzieho klúča takou, ktorá ešte neexistuje v tabuľke A ako primárny klúč. Súčasne sa ale vytvorí aj riadok v tabuľke A tak, aby naňho novozareadený riadok v tabuľke B referoval.

Tab. 4.1 Pravidlá referenčnej integrity

Popis referenčnej integrity môže byť zahrnutý aj v rámci ERD ako ohodnotenie prvkov reprezentujúcich relácie, viď. Obr. 4.14.



Obr. 4.14 Popis referenčnej integrity v ERD

Model podľa predošlého príkladu definuje zákaz rušenia tých výskytov entít A a C (napr. riadkov v príslušných tabuľkach zodpovedajúcej relačnej bázy), na ktoré referujú výskyt entít B, D a E (pravidlo DR). Pri zrušení výskytu entity D sa kaskádne zruší všetky výskytu entít E, ktoré na zrušený entitu referovali (pravidlo DC). Pri zrušení výskytu entity B sa vo všetkých výskytoch entity E, ktoré referovali na zrušený entitu B, nastaví referencia na "žiadnený výskyt B" - napr. *null* (pravidlo DN). Nový výskyt entity E môže mať nastavenú referenciu na "žiadnený" výskyt entity B (t.j. pri zaradení nového riadku do tabuľky zodpovedajúcej entite E sa nastaví cudzí klúč referujúci do tabuľky B na hodnotu *null*).

4.2.6. Tabuľkový popis fyzického dátového modelu

Pre detailný popis fyzického dátového modelu sa využívajú okrem entitno-relačných diagramov, diagramov dátových štruktúr a objektových diagramov aj detailné špecifikácie ďalejších vlastností entít, objektov a ich atribútov v **popisných tabuľkách**, napr. v štruktúrou a obsahom podľa Tab. 4.2. Informácie z takýchto popisných tabuľiek bývajú alebo by mali byť integrovanou súčasťou popisov prvkov projektovej databázy v CASE systémoch.

Plný názov dátovej entity	Identifikátor entity (tabuľky)					
Systémové názvy : subčne používané názvy v projekte alebo v systéme						
Štandardné doklady : odznačky na doklady, dokumenty, smernice, ktorých údaje sú v entite						
Alokácia: databázový server, databáza, súbor, ...						
Popis entity: popis s informáciami o význame, použití a obsahu danej entity						
Líšťa vlastností tabuľky - počet riadkov						
Maximum :	Primer:					
	Náraď (za časové obdobie):					
Vlastník:	kto je správcom, kto zodpovedá za aktuálnosť dát a pod.					
	Pristupové práva pre oprácie (R,I,U,D,A): aké má prístupové práva: R-ten čítanie, I-zaradovanie nových, U-zmena, D-rušenie, A-archivácia					
Fejllivateľ:	kto alebo čo využíva dátu					
ATRIBUT	LOGICKÝ TYP	FYZICKÝ TYP	KLÚČ	DOMÉNOVÁ INTEGRITA	REF. INTEGRITA PRE OPERÁCIE	REFERENCIA
názov	imprem. nezávislý	podľa impl. jazyka	(PK-primárny, FK-cudzí)	I U D		odkaz na id tabuľky – vážba na tabuľku

Tab. 4.2 Popisná tabuľka pre detailný popis entít

Pre modelovanie väzby dátových entít na funkcie (procesy, služby) sa používa tabuľka, ktorej záhlavie riadkov odpovedá entitám, záhlavie stĺpcov funkciám a položka pre daný riadok a stĺpec obsahuje informáciu o tom, aké operácie vykonáva funkcia v riadku nad entitou v riadku. Jedná sa o operácie čítania (R), vytvárania (I), aktualizácie (U), ničenia (D), archivácie (A), prípadne o ďalšie potrebné manipulácie s dátami.

ENTITA	FUNKCIA		
	fun1	fun2	...
E1	vykonávané operácie (R, I, U, D, A)		
E2			

Tab. 4.3 Tabuľka pre popis prístupu funkcií k dátovým entitám

Táto tabuľka sa označuje aj ako matica CRUD (podľa operácií Create, Read, Update, Delete). Poskytuje prehľad a využívaní dátových entít, o tom kde dátá vznikajú a kde sa likvidujú. Pomocou matice sa dajú lokalizovať principálne chyby v návrhu systému vynásávajúce z toho, že niektoré funkcie používajú dátá, ktoré žiadna funkcia nevytvára.

5. Súvislosti medzi analytickými modelmi

Rozšírený 3-rozmerný pohľad na systém pomocou dátového, funkčného a riadiaceho modelu pokrýva všetky kvalitatívne vlastnosti modelovaného systému podľa kap. 2.3. a je základom klasickej štruktúrovanej analýzy a návrhu. Hoci je možné jednotlivé modely vytvárať paralelne, obyčajne sa klade dôraz na dátový model (vzhľadom na jeho dlhšiu životnosť). Okrem toho je pri paralelnom vývoji potrebné zabezpečiť, aby nedošlo k divergencii modelov - vždy po určitých krokoch je potrebné všetky tri modely navzájom konfrontovať a zosúladiť na základe súvislostí medzi nimi.

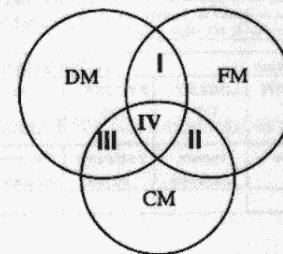
Súvislosti medzi modelmi vyplývajú zo základnej úlohy všetkých softvérových systémov – môžme ju formulovať nasledovne:

*spracovanie a/alebo poskytovanie informácií
v rôznej forme a rôznym spôsobom⁵.*

T.j. jadrom každého softvérového systému sú informácie a ich reprezentácia. Každý model systému sa týka informácií a ich reprezentácie na elementárnej alebo vyšszej úrovni – preto základom súvislostí – spoločným prienikom všetkých modelov sú informácie :

I. Súvislosti DM-FM:

- dátové toky vo FM súvisia s údajovými štruktúrami v DM
- dátové pamäti vo FM súvisia s údajovými štruktúrami v DM
- zapúzdrenie údajových štruktúr a funkcií vytvára triedy a objekty v zmysle objektovo orientovaného prístupu



II. Súvislosti FM-CM:

- interaktívne procesy vo FM súvisia so spôsobom riadenia interakcie definovanom v CM (v modele používateľského rozhrania)
- neinteraktívne procesy vo FM súvisia so spôsobom riadenia neinteraktívnej (automatickej) komunikácie – tento spôsob môže byť popísaný napr. modelom posielania správ v CM
- riadiace procesy vo FM súvisia so spôsobom generovania riadiacich signálov definovaným v rámci CM

III. Súvislosti DM-CM:

- definičný obor atribútov dátových entít v DM súvisia so spôsobom zabezpečenia zachovávania tohto definičného oboru a s reakciou na udalosti, ktoré predstavujú pokus o jeho prekročenie – spôsob zabezpečenia doménovej integrity
- relácie medzi dátovými entitami v DM súvisia so spôsobom zachovania integrity v reláciach – t.j. so spôsobom riešenie udalostí, ktoré by mohli viesť k narušeniu integrity

⁵ V širšom chápaniu sú informáiami aj riadiace signály – sú informáiami o spôsobe riadenia riadeného procesu. T.j. aj riadiace systémy spracovávajú a poskytujú informácie.

IV. Súvislosti DM-CM-FM:

- objekty danej triedy (spojenie DM a FM) majú definované správanie na základe riadenia prechodov medzi stavmi objektu, toto riadenie je definované v CM.

Súvislosti medzi modelmi sa prejavujú v ich **konzistencii** - nerozpornosti jednotlivých pohľadov. Konzistencia sa uplatňuje:

• V rámci jednoho modelu a tých istých metód a nástrojov

Napr. vo funkčnom modele v hierarchii diagramov dátových tokov (DFD) v jednotlivých dekompozíciah od kontextového diagramu až po triviálne procesy by malo byť zachované väzby dekomponovaného procesu k iným procesom, externým entitám a dátovým pamätiам realizované prostredníctvom dátových tokov - **zachovanie kontextu dekomponovaných prvkov**. Ak napríklad z procesu smeroval dátový tok do externej entity, musí aj v dekompozícii tohto procesu existovať dátový tok z procesu do tejto externej entity.

• V rámci jednoho modelu a rôznych metód a nástrojov

Napr. v dátovom modele existuje väzba medzi entitno-relačným diagramom (ERD) a stromovým diagramom dátových štruktúr (DSD) v modelovanom systéme. Listové položky DSD zodpovedajú elementárnym údajom a pokial nemajú dočasný charakter, mali byť buď reprezentované priamo nejakým atribútom entity z ERD alebo by mali byť z týchto atribútov vypočítateľné, odvoditeľné a pod.

• Medzi rôznymi modelmi

Napr. medzi procesným a dátovým modelom existujú väzby súvisiace s dátovými tokmi a pamäťami. Dátové pamäti v DFD môžu byť dekomponované do časti dátového modelu pomocou ERD. Dátové toky môžu byť dekomponované pomocou diagramov dátových štruktúr DSD. Pre dekompozíciu dátového toku a dátovej pamäti, do alebo z ktorého tok smeruje, platia väzby medzi DSD a ERD uvedené v predošлом bode.

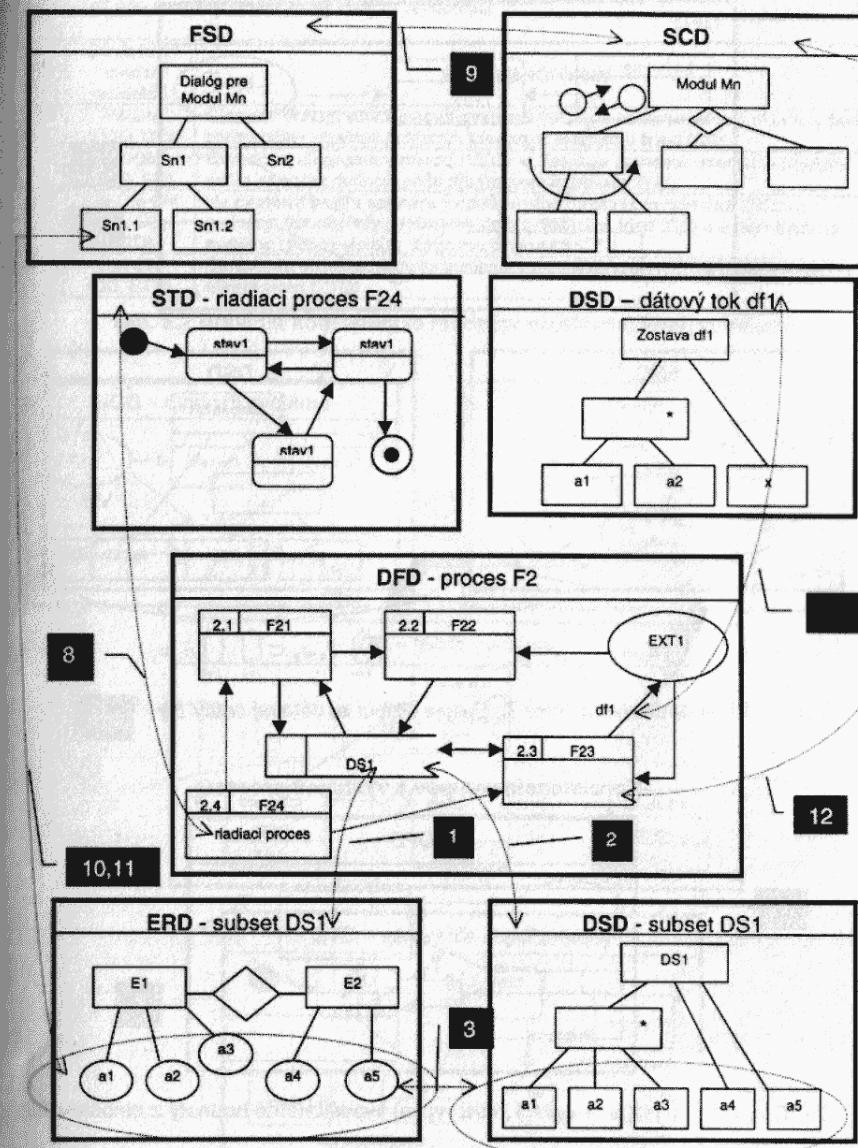
Zachovanie uvedených, prípadne ďalších konzistencií je možné zabezpečiť dodržiavaním pravidiel konzistencie modelov. Ich podpora je súčasťou dobrých CASE systémov. Nasledujúca tabuľka uvádzá prehľad možných súvislostí – konzistenčných pravidiel pre modely štruktúrovanej analýzy a návrhu.

Č.	Súvislosť medzi modelmi	Popis súvislostí
1.	FM-DM (DFD, ERD)	Dátová pamäť v DFD vo FM má dekompozíciu v ERD, ktorý odpovedá podmnožine alebo celej množine dátových entít ERD v DM (Obr. 5.1).
2.	FM-DM (DFD, DSD)	Dátová pamäť v DFD vo FM môže mať dekompozíciu v DSD, ktorý predstavuje model hierarchickej štruktúry celej bázy dát alebo jej časti. Tento DSD má v listových položkách len atribúty dátových entít DM (Obr. 5.1).
3.	FM-DM (DFD, ERD, DSD)	Ak má dátová pamäť v DFD dekompozíciu v DSD aj v ERD, potom množina listových položiek v tomto DSD je zhodná s množinou všetkých atribútov všetkých entít daného ERD (Obr. 5.1).
4.	FM-DM (DFD, DSD)	Dátový tok v DFD vo FM má dekompozíciu v DSD, ktorý predstavuje model hierarchickej štruktúry daného dátového toku (Obr. 5.1).
5.	FM-DM (DFD, DSD, ERD)	Dátový tok z procesu do dátovej pamäti v DFD môže v dekompozícii obsahovať len atribúty, ktoré patria dátové entité tvoriacej dekompozíciu danej dátovej pamäti (t.j. proces môže vkladať, meniť alebo rušiť len atribúty, ktoré patria entitám tvoriacim dekompozíciu danej dátovej pamäti) - konzistencia prístupov procesu k dátovej pamäti (Obr. 5.2).
6.	FM-DM (DFD, DSD, ERD)	Dátový tok z dátovej pamäti do procesu v DFD môže v dekompozícii obsahovať len atribúty, ktoré patria dátové entité tvoriacej dekompozíciu danej dátovej pamäti (t.j. proces môže čítať z dátovej pamäti len atribúty, ktoré patria

		entitám tvoriacim dekompozíciu danej dátovej pamäti) - konzistencia prístupov procesu k dátovej pamäti (Obr. 5.2).
7.	FM-DM (DFD, DSD, ERD)	Výstupný dátový tok procesu v DFD môže v dekompozícii obsahovať len atribúty vstupných tokov alebo hodnoty z nich vypočítateľné - konzistencia vstupov a výstupov procesu (Obr. 5.2).
8.	FM-CM (DFD, STD)	Riadiaci proces v DFD vo FM má dekompozíciu v STD, ktorý predstavuje stavový model generovania riadiacich signálov z riadiaceho procesu. Vstupné stavové informačné toky riadiaceho procesu v DFD zodpovedajú výstupným stavovo informačným premenným STD, na základe ich hodnôt a aktuálneho stavu sa vykonávajú prechody medzi stavmi v zodpovedajúcom STD. Výstupné riadiace toky riadiaceho procesu v DFD zodpovedajú generovaným výstupom STD (Obr. 5.1).
9.	FM-CM (SCD, FSD)	K interaktívemu modulu v SCD vo FM existuje model používateľského rozhrania – interakcie , napr. v tvare FSD (alebo STD).
10.	CM-DM (FSD, ERD)	Výstupné údajové polia, ktoré sú súčasťou nejakej interakčnej obrazovky v FSD (v modele používateľského rozhrania) a majú svoju reprezentáciu v báze dát, zodpovedajú atribútom dátového modelu (Obr. 5.1).
11.	CM-DM (FSD, ERD)	Výstupné údajové polia (zobrazované hodnoty), ktoré sú súčasťou interakčnej obrazovky v FSD (v modele používateľského rozhrania) a majú svoju reprezentáciu v báze dát, zodpovedajú atribútom dátového modelu alebo hodnotám, ktoré sú z týchto atribútov vypočítateľné (Obr. 5.1).
12.	FM-FM (DFD, SCD)	Proces v DFD má modulárnu štruktúru definovanú pomocou SCD. Ak je pomocou SCD dekomponovaný netriiválny proces, žiadene z procesov tvoriacich jeho dekompozíciu už nemá ďalšiu dekompozíciu do SCD (Obr. 5.1).

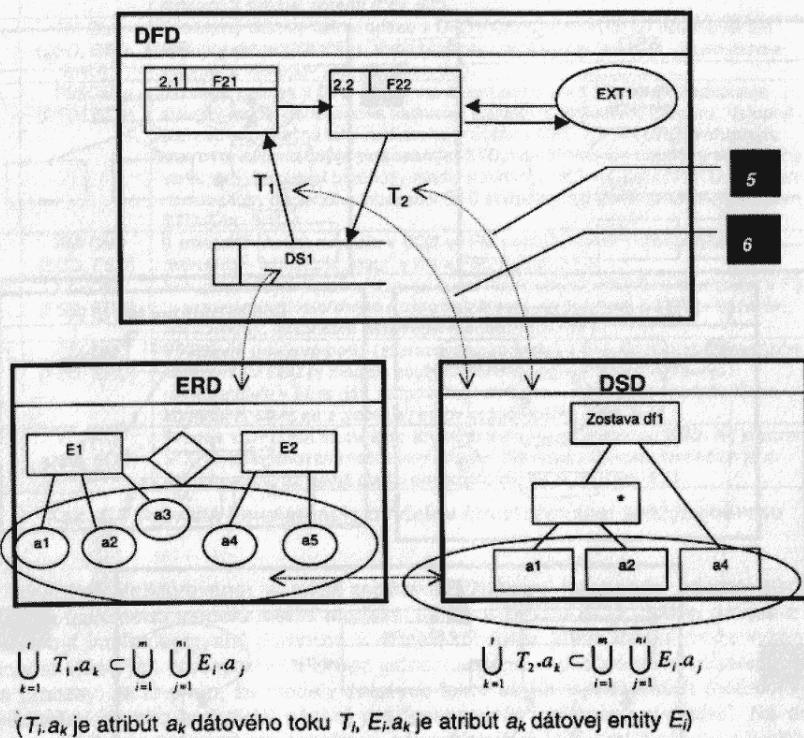
Tab. 5.1 Pravidlá konzistencie modelov štruktúrovanej analýzy/návrhu

V klasickej štruktúrovanej analýze a návrhu systémov neexistuje priama súvislosť v rámci funkčného modelu medzi modelmi na báze DFD a SCD. Jeden proces z DFD môže byť implementovaný pomocou viacerých modulov, jeden modul môže vykonávať činnosť viacerých procesov z DFD bez jednoznačného implicitného naviazania modulu na procesy. Je to preto, že modely dátových tokov slúžia v príslušných metodológiách predovšetkým ako analytický nástroj pre "pochopenie funkčnosti systému". Na druhej strane SCD sú nástrojom pre syntézu – návrh detailnej štruktúry systému z funkčného (modulárneho) pohľadu. Modely na báze DFD je ale možné použiť aj ako modely architektúry systému na báze podsystémov, skupín služieb a elementárnych služieb. Pri takomto upresnení sémantiky funkcií v DFD je možné použiť DFD ako model fyzickej štruktúry systému a detailný popis štruktúry služieb (ich algoritmu) vykonať pomocou SCD. Takáto súvislosť medzi DFD a SCD (konzistenčné pravidlo 12 v Tab. 5.1) vytvára nielen kompaktejšie modely, ale hlavne umožňuje hľadanie nekonzistencii a chýb v návrhu.

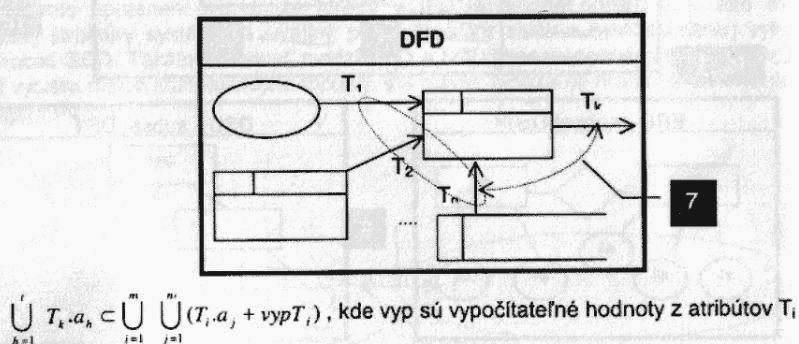


Obr. 5.1 Súvislosti medzi modelmi – časť I

Konzistencia prístupov procesu k dátovej pamäti



Konzistencia vstupov a výstupov procesu

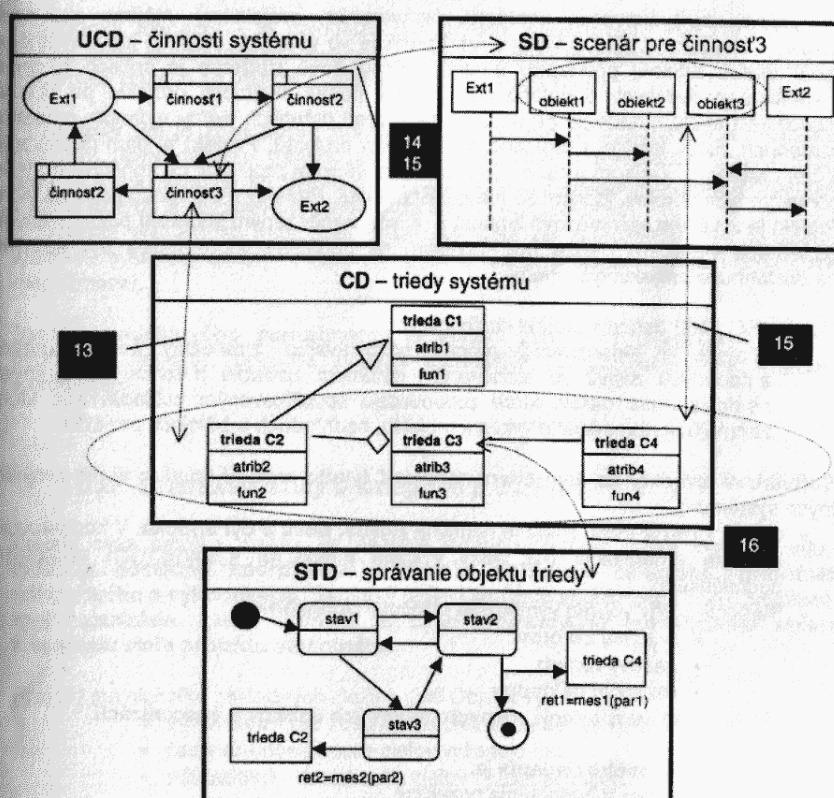


Obr. 5.2 Súvislosti medzi modelmi – časť II

Nasledujúca tabuľka uvádza prehľad niektorých možných súvislostí – konzistenčných pravidiel pre modely objektovej analýzy/návrhu.

Č.	Súvislosť medzi modelmi	Popis súvislostí
13.	FM-OM (UCD, CD)	Cinnosť v diagrame činností systému (UCD) je zabezpečovaná aktivitou danej podmnožiny objektov patriacich triedam z diagramu tried (CD).
14.	FM-CM (UCD, SD)	Cinnosť z diagramu činností (UCD) sa realizuje pomocou interakcie objektov podľa scenára definovaného diagramom scenárov (SD).
15.	CM-OM-FM (SD, CD, UCD)	Na činnosti podľa scenára modelovaného diagramom scenárov (SD) sa podieľajú len objekty patriace triedam z diagramu tried (CD) a externé entity systému (používateľia) z diagramu činnosti (UCD).
16.	OM-CM (CD, STD)	Správanie objektu triedy definovanej v diagrame tried (CD) je dané stavovým diagramom (STD).

Tab. 5.2 Pravidlá konzistencie modelov objektovej analýzy/návrhu



Obr. 5.3 Súvislosti medzi modelmi – časť III

6. Modelovanie v riadení softvérových projektov

Projektové riadenie (project management, projektový manažment, riadenie projektov) je disciplína, ktorá sleduje väzby medzi ekonomikou a technológiou, informačné technológie nevyňímajúc. Z tohto dôvodu sa táto časť zaobrázá základnými pojмami, používanými modelmi a všeobecne platnými princípmi projektového riadenia. Projektové riadenie sa uplatňuje na softvérový systém v etapách jeho analýzy, návrhu a implementácie. Modely, ktoré sa používajú v projektovom riadení, súvisia so vzťahom projektu a jeho okolia, s používanými postupmi pri riadení a koordinácii (modelovanie aktivít a úloh), so štruktúrou a väzbami medzi zdrojmi (modely štruktúry realizačných tímov a komunikácie v nich).

6.1. Základné pojmy v projektovom riadení

Projektom softvérového systému (softvérovým projektom) môžeme rozumieť softvérový systém a jeho súvisiace reprezentácie vo všetkých etapách jeho životného cyklu okrem rutinnej prevádzky a údržby. **Projektové riadenie** je spôsob riadenia a koordinácie ľudských a materiálnych zdrojov počas životnosti projektu pri použití moderných techník riadenia na dosiahnutie vopred určených cieľov v danom rozsahu, nákladoch, čase, kvalite a spokojnosti účastníkov projektu. **Projekt** je druh podnikania s definovaným začiatkom a cieľmi. V praxi závisí väčšina projektov od konečných alebo obmedzených zdrojov, ktorími sa má dosiahnuť cieľ. Projekty majú formu riešenia úloh. Projekt je zvyčajne jednorázová činnosť s presne vymedzeným súborom požadovaných konečných výsledkov. Môže byť rozdelený na podúlohy, ktoré musia byť splnené na dosiahnutie projektových cieľov.

Norma ISO 8402 definuje projekt nasledovne:

Projekt je jednoznačný proces, pozostávajúci z množiny koordinovaných a riadených aktivít so stanoveným dátumom začiatku a konca, zameraných na dosiahnutie cieľov, ktoré zodpovedajú špecifikovaným požiadavkám, ktoré zahrňujú aj obmedzenia v trvaní projektu, nadhľadoch a zdrojoch projektu.

Softvérové projekty sa dajú charakterizovať týmito znakmi (platí to aj pre projekty iných systémov):

- Úloha, ktorú je treba splniť je relativne zložitá, nová a dynamická. V hodnotovom systéme organizácie má veľký význam a rieši sa v jedinečných situačných podmienkach.
- Riešenie úlohy vopred vymedzujú rámcové podmienky:
 - použitie zdrojov
 - časový rozvrh
 - rozpočet nákladov
- Úloha vyžaduje zapojenie rôznych odborných úsekov a špecializácií.

Obsahom projektového riadenia je:

1. výber a hodnotenie projektov
2. príprava projektov
3. realizácia projektov

Projektový manažment znamená odklon od manažmentu individuálnych riešení, opakujúcich sa činností, pri ktorých je možná značná predikovateľnosť. Je to odklon od determinovanosti a určitosti i od klasických organizačných štruktúr. Týmto sa odlišuje projektový manažment od tzv. **procedurálneho manažmentu**, kde sú situácie viac-menej predikovateľné, menej riskantné a stabilné. Procedurálny manažment má tendenciu k centralizovanému rozhodovaniu a striknému lipnutiu na hierarchii autorít. Nové technológie a zmeny na trhu vyvolávajúce potrebu väčšej adaptability spôsobujú nutnosť vyššieho stupňa technických a manažérskych schopností a decentralizácie rozhodovania.

Projektový manažment sa odlišuje od tradičných foriem manažmentu v niekoľkých úrovnach daných existenciou troch **základných faktorov projektového manažmentu**:

- projektový manažér
- projektový tím
- systém projektového manažmentu.

Projektový manažér - je najdôležitejším prvkom projektového manažmentu. Je to osoba, ktorá je individuálne zodpovedná za integráciu pracovného úsilia jednotlivých odborných činností za účelom dosiahnutia stanovených cieľov.

Projektový tím je zoskupenie individualít a podskupín do koherentného pracovného tímu pre účely riešenia daného projektu. Vytvorenie projektového tímu je zložité, pretože v tejto skupine sa stretávajú ľudia z rôznych funkčných oblastí a organizácií, v rámci ktorých je požadovaná participácia na rôznych úrovnach. Z hľadiska efektívnosti musí mať projektový manažér a projektový tím k dispozícii efektívny systém pravidiel vedúcich k splnenie cieľov projektu - systém projektového manažmentu.

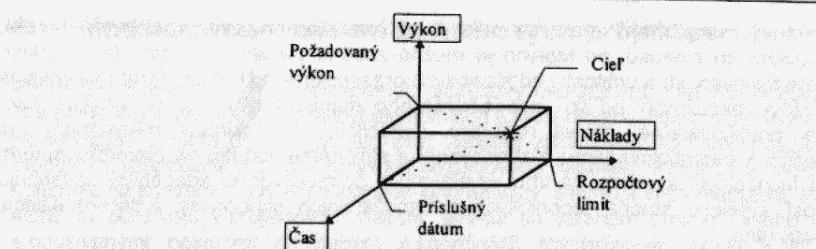
Systém projektového manažmentu je systém zahrňujúci organizačnú štruktúru, informačné procesy, pravidlá, praktiky a procedúry umožňujúce riešenie projektu v organizácii.

6.2. Charakteristiky a kategórie projektov

Projekt býva natoľko zložitý, že riešenie jednotlivých úloh, ktoré vedú k cieľu, si vyžaduje dôslednú koordináciu a riadenie z hľadiska časového, prioritného, nákladového a výkonového. Samotný projekt sa často koordinuje s inými projektmi tej istej organizácie. z toho vyplýva, že ciele projektového manažmentu smerujú k splneniu cieľa projektu samotného.

Projekt má niekoľko základných zložiek (vid.Obr. 6.1):

- výkonový - pre dosiahnutie cieľov musia byť vykonané určité činnosti
- časový - dosiahnutie cieľov vyžaduje čas
- nákladový - dosiahnutie cieľov vyžaduje financie



Obr. 6.1 Výkonné, nákladové a časové ciele projektov

Charakteristiky aktivít spojených s vypracovaním projektu:

- Účelovosť** - Projekt zahrňuje jednotlivé, účelovo definované ciele, ukončené výrobkom alebo iným výsledkom. Zvyčajne sú tieto ciele špecifikované časovými nákladmi, postupom a požiadavkami.
- Interdisciplinárnosť** - Projekty sa neriešia v pôvodných organizačných liniách, pretože potrebujú využiť vedomosti a talent z **interdisciplinárnych oblastí** a organizácií. Komplexnosť projektov vzniká často z komplexnosti výhod technológií, ktoré participujú na projektových prácach a môžu priniesť nové, unikátné problémy.
- Unikátnosť** (jedinečnosť, neopakovateľnosť) - Každý projekt je unikátny a požiadavky kladené na konkrétny projekt sú špecifické. Ak vychádzame z predchádzajúceho bodu, potom každý nový projekt potrebuje novú technológiu, postupy, a pre organizáciu realizujúcu projekt predstavuje faktor neistoty a rizika.
- Dočasnosť** - Projekty sú **dočasné aktivity**. Organizovanie personálu, materiálu a príležitostí je komplikovaný cieľ zvyčajne stanovený časovým rámcom. Ak je tento cieľ úspešne dosiahnutý, organizácia je rozpuštená alebo rekonfigurovaná pre prácu na nových projektoch.
- Viacfázovosť** - Projekt je proces pre úspešné dosiahnutie stanoveného cieľa. Počas tohto procesu prechádza projekt niekoľkými fázami, ktoré sa nazývajú **životný cyklus projektu**. Úlohy, ľudia, organizácia a ostatné zdroje spôsobujú, že sa projekt posúva z jednej etapy do ďalšej. Organizačná štruktúra a výdavky na zdroje spolu s priebehom jednotlivých fáz signalizujú mieru kompletizácie projektu, resp. jeho ukončenia.

Projekt má špecifický **začiatok, cieľ, účel a ukončenie**, musí mať pridelené špecifické **prostriedky**. Obyčajne je určený pre použitie inými osobami.

Vývoj novej softvérovej aplikácie patrí do kategórie tzv. **problémových projektov** (okrem toho poznáme ešte napr. investičné a inovačné projekty).

Problémový projekt má nasledujúce charakteristiky:

- Organizačná forma** - najčastejšie vyhovuje tímová organizácia, ktorá je najmenej náročnou formou, pretože nevyžaduje zásah do stabilnej organizačnej štruktúry (nemusí to však, hlavne v prípade implementačnej etapy vývoja softvérového systému, plati).
- Tažisko projektu** - spočíva v riešiteľskej činnosti.
- Štrukturalizácia projektu** - projekt sa člení - fázuje zvyčajne odborne (v prípade softvérových projektov v súlade so **životným cyklom programového vybavenia**).

- Vedúci projektu** má na starosti hlavne časovú a vecnú koordináciu práce špecialistov, prijíma potrebné rozhodnutia a opatrenia.
- Plánovanie a kontrola** prebieha na úrovni časových plánov plnenia dielčích úloh.
- Optimalizácia** sa uplatňuje v rôznych variantoch, z ktorých sa volí najvhodnejšie riešenie z hľadiska cieľov.

Ciele projektového manažmentu sú jednoznačne dané **cieľmi projektu**. Teda dosiahnutie cieľov projektového manažmentu pri konkrétnom projekte (uriadenie projektu do úspešného konca) znamená aj dosiahnutie cieľov projektu (vytvorenie konkrétnej aplikácie a pod.).

6.3. Predpoklady úspechu projektového riadenia

Pre dosiahnutie cieľov projektu je potrebné zabezpečiť všetkými dostupnými prostriedkami podmienky nevyhnutné pre úspech tohto projektu. Pre tieto účely je dôležité zohľadniť nasledujúce faktory:

- Funkcia iniciátorov** - Ten, kto zakladá projektovú skupinu, musí s ňou udržiavať stály kontakt a poskytovať potrebnú pomoc a podporu. Musí presadzovať schopnosti a uznanáť autonómiu skupiny. Na druhej strane musí zasahovať, ak sa vyskytnú chyby v rozhodovaní.
- Príprava kooperacie** - Je potrebné sa rozhodnúť, ktorých pracovníkov vybrať a koľko času majú tito venovať prácam na projekte. Toto je veľmi citlivá otázka a od jej úspešného vyriešenia závisí do veľkej miery úspech projektu.
- Tvorba projektovej skupiny** - Skupina je v značnej mieri závislá od stanovených cieľov projektu. Počas tohto procesu musia byť zohľadnené individuálne prednosti každého člena z hľadiska špecifickosti projektu. Predovšetkým sa musia brať do úvahy dva faktory: veľkosť skupiny a kvalifikácia jednotlivých jej členov.
- Role** - Značnou výhodou projektovej skupiny je, ak sa jej členovia dokážu stotožniť s jednotlivými rolami, ktoré sa vyvíjajú v každej skupine. Sú to role: vedúceho, aktivizátora, tvrdohlavca, analyтика, pedanta, lojalného pracovníka a poradcu.
- Kontrola napredovania** - Túto kontrolu zabezpečuje iniciátor v úzkej koordinácii s vedúcim projektu, tak po kvalitatívnej ako aj po kvantitatívnej stránke.
- Konečné termíny** - Ukončenie projektovej úlohy je potrebné presne časovo vymedziť. Ak je toto rozhodnutie odkladané, chýba, alebo sa porušuje, vplýva to negatívne na pracovnú skupinu i na iniciátora.

Projekt je zvyčajne **jednorázová akcia** vyžadujúca dačasné **zvláštne organizačné usporiadanie**. S riešením každého projektu je spojené značné riziko. Príprava projektu potrebuje **mimoriadne opatrenia**, ktoré vyžadujú čas a náklady. Preto **neúspech projektu** môže mať **závažné dôsledky** na všetkých zúčastnených:

- finančné a materiálne straty** - strata času a všetkých nákladov spojených s projektom
- škody na prestíži** zúčastnených vedúcich a výkonných pracovníkov.

Za týchto podmienok môže byť strach z neúspechu projektu bariérou angažovanosti a efektívnosti projektu. Existujú postupy smerujúce ku značnému zníženiu rizika neúspechu a zaisteniu úspešnej realizácie projektu:

1. **Koncept (štúdia realizovateľnosti, feasibility study).** Koncept ako prípravná fáza projektovania musí dať istotu, že hospodárske a technické ciele projektu budú dosiahnuté.
2. **Závažnosť zámeru.** Niektoré veľké a nákladné zámery sa dajú uskutočniť rutinným spôsobom s danými odbornými znalosťami. Prípadné chyby sú malé a známe. Riziko škôd je v tomto prípade pomerne malé. Inak je to pri zámeroch, ktoré sa nedajú zvládnuť rutinným spôsobom. Vzniká problém, keď doterajšia rutinná organizácia nemôže zabezpečiť požadovanú istotu konceptu. Preto je potrebné pri námete projektu zvážiť jeho dosah.
3. **Metodický systém.** Metodickú základňu projektu tvorí organizácia výstavby projektu a organizácia priebehu projektu. Z hľadiska zabránenia neúspechu má najväčší význam budovanie a fungovanie systému plánovania a kontroly termínov.
4. **Kontrola správneho myšlenia** spočíva v bežnom preskúšavaní úvah projektovej skupiny a v dispozících, ktoré uskutočňuje. V podstate ide o odstránenie chýb, ktoré sa bežne vyskytujú pri tvorivom myšlení. Funkciu kontrolóra môže vykonávať kvalifikovaný člen projektovej skupiny alebo externý expert.
5. **Vedúci projektu** je najvýznamnejším činiteľom pre úspešné zvládnutie projektu. Jeho funkcia sa odlišuje tak od bežného typu vedúceho pracovníka, ako aj od špecialistu. Čažisko jeho kvalifikácie nespočíva vo vecnej odbornosti, ale v schopnostiach a v povahových rysoch. Ide hlavne o organizačné schopnosti a o umenie viesť ľudí. **Charakteristické vlastnosti vedúceho projektu** by mali byť tieto:
 - rozvíja iniciatívu a je pripravený robiť rozhodnutia
 - má pozitívne zameranie k tímovej práci a môže motivovať spolupracovníkov
 - preukazuje obratnosť v jednaní a riadení
 - má schopnosť vciť sa
 - dobre pozná podnik.
6. **Tvorivá pracovná klíma.** Úspešne uriaditi projekt znamená riešiť nové problémy s vyššou efektívnosťou - tvorivým spôsobom. K tomu je potrebné uplatňovať zásady tímovej práce, kde sú osvedčili hlavne tieto postupy:
 - nepokúšať sa vyvíjať projekty orientované na tvorivosť pomocou bežnej organizácie a tradičnými tvorivými postupmi
 - využívať vedomosti a schopnosti pracovníkov v rámci efektívnej organizačnej formy
 - žiadna organizácia nefunguje sama
 - členov projektovej skupiny je potrebné vopred pripraviť na túto prácu.

6.4. Modely organizácie projektu

K zabezpečeniu dosiahnutia cieľov projektu je potrebná vhodná organizácia, zladenie tímovej práce na projekte a kvalitné metódy kontroly projektu. Ďalšie kapitoly budú v krátkosti pojednávať práve o tejto problematike.

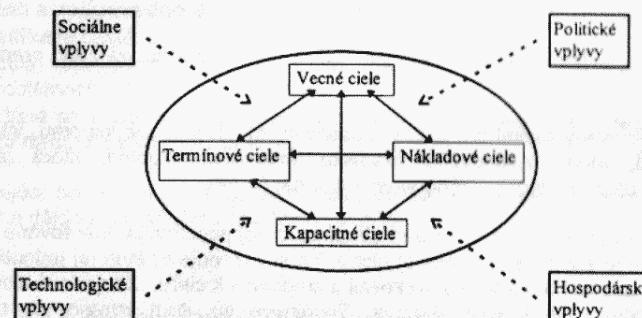
Tradičné princípy organizácie (ako už bolo uvedené vyššie) sú vzhladom na špecifický charakter projektového manažmentu nahradzované inými, vhodnejšími princípmi. Vznikajú teda pružné organizačné formy a tieto majú slúžiť dvom cieľom:

1. organizácia musí rýchlo a trvalo reagovať na zmeny prostredia.
2. pri riešení komplexných problémov musí čo najefektívnejšie využívať špecifické poznatky a schopnosti členov organizácie.

Projekt ako taký možno vymedziť ako **systém**, ktorého elementy sú:

- **vecné ciele**
- **termínové ciele**
- **kapacitné ciele**
- **nákladové ciele**

Na projekt ako systém z toho najväčšejšieho hľadiska pôsobia rôzne vplyvy (Vid.Obr. 6.1).



Obr. 6.1 Projekt ako systém

V praxi sa osvedčila **projektová a maticová organizácia projektu**, ktoré zabezpečujú nielen vertikálne, ale aj horizontálne prepojenie pri riadení projektu. **Vertikálne vzťahy** známe z línia-funkčnej (štábejnej) organizačnej štruktúry ostávajú zachované, kým novo sa vytvárajú **horizontálne vzťahy** medzi účastníkmi projektu na jednej úrovni. Tieto vzťahy má na starosti vedúci projektu a ten je aj podmienkou pružnosti takejto architektúry.

Pružné organizačné štruktúry sa podľa kritéria viacnásobnej podriadenosti pracovníkov a časového trvania rozlišujú na:

- **Projektovú organizáciu projektu:**

V prípade projektovej organizácie sú rozhodovacie právomoci čiastočne decentralizované a prenesené na nižšie zložky riadenia. Vznikajú tým odborné miesta s ohrianičenými úlohami. Je tu dôležité miesto projektového koordinátora.

- Maticovú organizáciu projektu:**

Je typickejšia pre vývoj programového vybavenia, ako projektová organizácia. Línie práv a povinností prebiehajú tak vo vertikálnom, ako aj v diagonálnom smere. Táto organizácia narúša princíp „jeden človek - jeden vedúci“. V dôsledku tohto sa treba sústrediť na rozpracovanie vzťahu vedúci projektu s vedúcimi útvarov a viest' si evidenciu pracovného času špecialistov na projekte, túto plánovať a pod.

Vo všeobecnom prípade sa každý vykonávateľ zodpovedá niekomu za svoju odbornosť, ako aj za splnenie cieľov a úloh projektu.

6.5. Tímová práca

Skupina sú dve alebo viacej osôb, ktoré sa nachádzajú v týchto podmienkach:

1. vzťahy medzi členmi skupiny sú navzájom závislé, teda správanie sa každého člena ovplyvňuje správanie ostatných členov skupiny.
2. členovia majú spoločnú ideológiu - súbor názorov, hodnôt a noriem, ktoré regulujú ich vzájomné vystupovanie

Kolektív je definovaný ako skupina, ktorá má spoločné záujmy predpokladajúce jednotu cieľov.

Tím je definovaný rozsahom cieľov a časovým hľadiskom pre skupinu. Väčšinou je to krátkodobá, alebo z časového hľadiska limitovaná skupina, ktorá je vytvorená za účelom plnenia presne definovanej aktuálnej úlohy.

Pracovná skupina je organizovaná skupina, ktorej poslanie je orientované na základný cieľ, ktorým je splnenie stanovenej úlohy. To si vyžaduje prístup jej jednotlivých členov tak, aby v súhrne navzájom nadvázovali a smerovali k cieľu. Skupinové ciele sa stávajú cieľmi každého člena tejto skupiny. Skupinové sú v tom zmysle, že sa v skupine vytvárajú a v jej rámci sú uskutočňované.

Medzi faktory ovplyvňujúce prijatie skupinového cieľa jednotlivými členmi skupiny patria:

- vnímanie súvislostí medzi skupinovými cieľmi a vlastnými potrebami jej členov
- jasnosť a zrozumiteľnosť skupinových cieľov
- vysoká súdržnosť skupiny
- účasť členov pri stanovení cieľov
- dobrá akcieschopnosť skupiny

Plnenie úloh skupiny, t.j. dosahovanie cieľov, výkon a udržiavanie skupiny a jej stability v správaní sa jej členov, je podmienené **normami skupiny**. Tie sa definujú ako **vzory očakávaného alebo požadovaného správania** sa, pričom v pracovnej skupine pôsobia ako nepísané, skupinou prijaté a väčšinou skupiny dodržiavané pravidlá správania sa.

Tímová práca má zmysel hlavne pri plnení komplexných úloh. Vo všeobecnosti platí, že **duševný potenciál skupiny prevyšuje jednoduchú sumáciu možností jej**

jednotlivých členov. Taj je tomu napríklad v prípade vývoja programového vybavenia pre špecifické účely, keď do projektu vkladajú svoje vedomosti tak špecialisti z oblasti, pre ktorú je programové vybavenie určené, ako aj programátori, ktorých vedomosti sú hlavne z oblasti programovania.

Znaky dobre fungujúceho pracovného tímu smerujúceho k dosiahnutiu cieľov projektu sú:

- Atmosféra v tíme má sklon k **neformálnosti**, uvoľneniu.
- V tíme sa veľa **diskutuje**, diskusie sa zúčastňuje každý a vždy sa týka úlohy tímu.
- Úlohu alebo cieľ tímu **dobre chápú a prijímajú** ostatní členovia. K určitej časti cieľa sa rozvíja voľná diskusia.
- Členovia tímu sa navzájom **akceptujú a vypočujú**. Vypočuje sa každý nápad, aj keď sa javí ako nezmyselný.
- Nie je absencia nesúhlasu, pri jeho prejavoch sa skúmajú dôvody nesúhlasu bez snahy o ovládnutie nesúhlasiaceho.
- Rozhodnutia sa väčšinou dosahujú cestou **všeobecnej dohody**.
- Všetci stoja za prijatými rozhodnutiami. Je zreteľné a akceptované **rozdelenie úloh a zodpovednosti**.
- **Kritika** je častá, úprimná a pomerne pokojná.
- Ľudia **slobodne** vyjadrujú svoje nápady a dojmy tak o jednotlivých problémoch, ako aj o úknoch tímu.
- V tíme sa vodcovstvo presúva z času na čas podľa okolnosti. **Otázkou nie je, kto riadi, ale ako robíť prácu.**

Takto fungujúci tím by mal byť zárukou toho, že sa splnia ciele a úlohy, súvisiace s projektom a dôjde k úspešnému zavŕšeniu projektu.

6.5.1. Modely organizačnej štruktúry softvérového tímu

Ak je riešením projektu poverených viac ľudí, na úroveň produktivity práce a kvalitu výsledného produktu má zásadný vplyv, okrem zručnosti a schopnosti jednotlivých členov, **organizačná štruktúra tímu**.

Existujú tri základné modely organizačnej štruktúry tímu:

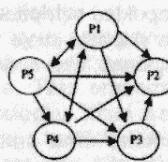
1. demokratická organizácia (ego-less team)
2. hierarchická organizácia (chief programmer team)
3. riadená decentralizovaná (controlled decentralized team)

Demokratická organizácia tímu

Charakteristické znaky:

- počet členov menší alebo rovný 10
- ciele tímu sú určené dohodou
- pri zásadných rozhodnutiach majú rovnakú váhu priponenky každého člena
- úloha vedúceho tímu rotuje medzi členmi
- veľké interpersonálne komunikácie medzi členmi





Obr. 6.2 Interpersonálne komunikácie v demokratickom tíme

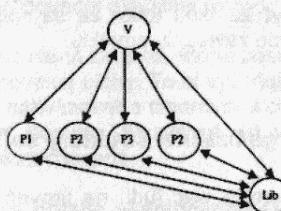
Využitie:

- pre dlhodobé, časovo neterminované výskumné typy projektov (pre bežné, nie príliš všeobecné úlohy, časovo terminované je táto organizácia tímu malo efektívna)

Hierarchická organizácia tímu

Charakteristické znaky:

- vedúci programátor je zodpovedný za všetky dôležité rozhodnutia, robí návrh projektu, rozdeľuje prácu medzi programátorov
- vedúcomu pomáha pri rozhodnutiach, prípadne ho zastupuje programátor v zálohe (back-up programmer)
- za dokumentáciu je zodpovedný knihovník (librarian)
- obmedzené interpersonálne komunikácie medzi členmi



Obr. 6.3 Interpersonálne komunikácie v hierarchickom tíme

Využitie:

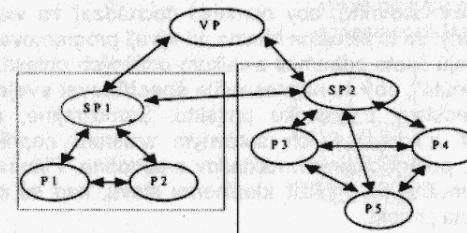
- pre projekty s jednoduchým riešením a presným termínom ukončenia

Riadená decentralizovaná organizácia tímu

Charakteristické znaky:

- kombinácia demokratickej a hierarchickej organizácie
- vedúci programátor, VP je zodpovedný za všetky dôležité rozhodnutia, robí návrh projektu, rozdeľuje prácu medzi programátorov
- starší programátori (SP) riadia jednotlivé podskupiny, pomáhajú vedúcomu pri rozhodnutiach
- v jednotlivých podskupinách je demokratická organizácia

- menej interpersonálnych komunikácií ako v demokratickom ale viac ako v hierarchickom tíme



Obr. 6.4 Interpersonálne komunikácie v riadenom decentralizovanom tíme

Využitie:

- pre rozsiahle projekty, ktoré nie sú veľmi komplikované

6.6. Plánovanie a kontrola v projektovom riadení

Tak ako každý zložitý problém, aj projekt je potrebné rozčleniť do jednotlivých fáz s uvedením ich vecného obsahu, nositeľov, dĺžky trvania, cieľov, prípadne nákladov. Softvérový projekt sa fázuje v súlade s **životným cyklom programového systému**, vid. kapitola 0). Zároveň treba zohľadňovať činitele, akými sú prirodzená štruktúra projektu, súbor dielčích cieľov, požiadavky na odbornosť pracovníkov, organizačné vzťahy, časové a priestorové nároky a podobne. **Fáza** by mala byť organizačne, odborne, metodicky a časovo ucelená časť projektu s jasným cieľom a konkrétnymi stanovenými výsledkami, ktoré sa majú dosiahnuť v danej etape. **Účelné rozčlenenie** do fáz môže podstatne ovplyvniť priebeh aj výsledky projektu.

Priebeh riadenia projektu je vždy spojený s vysokou mierou **neistoty a rizika**. Je to proces iteratívny a spätnoväzobný. Poznatky a závery, ku ktorým sa dospeje v určitej fáze, sú predpokladom pre ich zhodnotenie a porovnanie s plánom určujúcim ciele projektu. Tieto závery môžu slúžiť na úpravy, doplnky a zmeny dosiahnutého stavu, alebo sú východiskom pre presnejšie ponímanie a plánovanie ďalšej fázy. To umožňuje bezprostredne prihliadať ku zmeneným podmienkam alebo chybám a okamžite prijímať príslušné opatrenia. Tým sa podstatne znižuje riziko neúspechu.

Čas po ukončení jednej fázy je vhodný pre:

1. vyhodnotenie doterajšieho a očakávaného priebehu
2. uskutočnenie strategických rozhodnutí.

V tej najväčšej rovine spočíva riadenie projektov v plánovaní, realizácii a kontrole projektov.

V súvislosti s cieľmi projektu (a teda aj cieľom projektového manažmentu - uradiť projekt k vyriešeniu problému a splneniu cieľov projektu) vystupuje do popredia potreba presnej špecifikácie cieľov projektu **záklazníkom**. Potom je dôležité v prvom rade presné vymedzenie pojmov (slovník), aby nemohlo dochádzať ku viacznačnostiam z jednej alebo z druhej strany. Je to aktuálne hlavne pri vývoji programového vybavenia, kde pri analýze sa stretávajú často odborníci z celkom odlišných oblastí. Pri zadávaní cieľov treba zákazníka „donútiť“, aby čo najpresnejšie špecifikoval svoje požiadavky, presne vyjadril svoje predstavy o výsledku projektu. Samozrejme, analytik tieto predstavy musí podchýtiť a korigovať ich správnym smerom napríklad v zmysle reálnosti ich uskutočnenia, potencionálnych nákladov a podobne. Plánovanie cieľov je potom jednoduchšie a je možné sa priblížiť ideálному stavu, keď sú obom stranám jasné požiadavky kladené na projekt.

6.7. Metódy a nástroje projektového riadenia

Pre riadenie projektov sa používajú metódy používajúce rôzne modelovacie nástroje. Existujú 3 základné oblasti, pre použitie modelov v riadení projektov:

1. odhady ceny realizácie projektu
času potrebného na realizáciu projektu
počtu pracovníkov potrebných pre realizáciu projektu
2. aktualizácia a kontrola týchto odhadov
3. plánovanie, riadenie a kontrola úloh a činností ľudí zúčastňujúcich sa na projekte

6.7.1. Odhad pracnosti softvérových systémov

Cena softvérového projektu je závislá na pracnosti projektu (udáva sa v človekomesiacoch - person-month PM). Pre odhad ceny sa používajú modely s jednou premennou (jednoparametrické modely) a viacparametrové modely (CoCoMo). CoCoMo (Constructive Cost Model) je Boehmov empirický cenový model [4] pre určenie ceny projektu na základe známeho alebo dobre odhadnutého rozsahu programu. CoCoMo poskytuje tri úrovne modelov s rastúcou úrovňou zložitosti (základný, stredný, detailný). Ďalší popis sa týka CoCoMo strednej úrovne.

Základné kroky zostavenia cenového modelu (modelu pracnosti) CoCoMo:

I. Získanie počiatočného - nominálneho odhadu pracnosti na základe odhadu dĺžky kódu

$$En = a * (KDL)^b \quad [PM]$$

a odhadu doby trvania (v mesiacoch - M)

$$t = c * E^d \quad [M]$$

- En (effort) je nominálny odhad pracnosti v človekomesiacoch [PM]

- KDL (kilo delivered lines) je odhad rozsahu projektu v tisícoch riadkov zdrojového textu
- a, b, c, d sú empirické konštanty závislé od typu projektu
- t je odhad doby trvania projektu v mesiacoch [M]

TYP PROJEKTU	a	b	c	d
organický	3.2	1.05	2.5	0.38
polouzavretý	3.0	1.12	2.5	0.35
uzavretý	2.8	1.20	2.5	0.32

Typy projektov podľa CoCoMo:

TYP PROJEKTU (MÓD)	POPIŠ
Organický (organic)	Projekt v oblasti, v ktorej má riešiteľská organizácia značné skúsenosti, požiadavky na projekt sú menej prísne, rieši sa obyčajne malým tímom (napr. projekt jednoduchých obchodných systémov a systémov na spracovanie údajov).
Uzavretý - viazaný (embeded)	Projekt v novej oblasti, v ktorej má riešiteľská organizácia malé skúsenosti. Na projektovaný systém sú kladené prísne požiadavky z hľadiska bezpečnosti, kvality rozhrania a pod., systémy musia zohľadňovať obmedzenia prostredia dané hardvérovými, softvérovými a personálnymi zložkami. Príkladom sú projekty informačných a riadiacich systémov pre riadenie letovej prevádzky, riadenie technológie výroby a pod.
Polouzavretý - prechodný (semidetached)	Sú projekty, ktoré sa vzhľadom na svoju zložitosť, rozsah, obmedzenia nachádzajú medzi organickými a uzavretými projektami.

II. Určenie 15 násobiacich CoCoMo faktorov k a výsledného upravujúceho faktoru EAF (Effort Adjustment Factor)

$$EAF = k_1 * k_2 * \dots * k_{15}$$

Úroveň hodnoty							
Koeficient	Popis	VL	L	N	H	VH	EH
k1 RELY	Požadovaná spoľahlivosť projektu	0.75	0.88	1.00	1.15	1.40	
k2 DATA	Rozsah údajovej bázy		0.94	1.00	1.08	1.16	
k3 CPLX	Zložitosť a komplexnosť	0.70	0.85	1.00	1.15	1.30	1.65
k4 TIME	Časové obmedzenia (požiadavky na dobu odozvy)			1.00	1.11	1.30	1.66
k5 STOR	Pamäťová náročnosť			1.00	1.06	1.21	1.56
k6 VIRT	Stabilita počítača (hotový alebo vyvíjaný firmware)		0.87	1.00	1.15	1.30	
k7 TURN	Rýchlosť odozvy počítača pri vývoji		0.87	1.00	1.07	1.15	
k8 ACAP	Znalosti a skúsenosti analytikov	1.46	1.19	1.00	0.86	0.71	
k9 AEXP	Skúsenosti s aplikáciou	1.29	1.13	1.00	0.91	0.82	
k10 PCAP	Skúsenosti a schopnosti programátorov	1.42	1.17	1.00	0.86	0.70	
k11 VEXP	Znalosti virtuálneho počítača (operačný systém, programové prostredie)	1.21	1.10	1.00	0.90		

k12	LEXP	Znalosti o programovacích jazykoch	1.14	1.07	1.00	0.95		
k13	MODP	Moderné programovacie techniky	1.24	1.10	1.00	0.91	0.82	
k14	TOOL	Použitie SW nástrojov	1.24	1.10	1.00	0.91	0.83	
k15	SCED	Časový plán (voľný alebo napäť)	1.23	1.08	1.00	1.04	1.10	

Úrovne hodnôt koeficientov:

VL	veľmi nízka
L	nízka
N	nominálna
H	vysoká
VH	veľmi vysoká
EH	extra vysoká

III. Výpočet upraveného odhadu pracnosti: $E = EAF \cdot En$ [PM]

6.7.1.1. Rozdelenie pracnosti projektu do fáz

Pre plánovanie projektu je potrebné odhadovať **pracnosť jednotlivých fáz riešenia**. v CoCoMo je pracnosť jednotlivých fáz vyjadrená percentuálnym podielom z celkovej pracnosti. Tento podiel je závislý od typu a veľkosti projektu. Pre **organický typ projektu** je rozdelenie pracnosti nasledujúce:

Celkový odhadovaný rozsah projektu

Fáza	2 KDL	8 KDL	32 KDL	128 KDL
Návrh	16	16	16	16
Detailný návrh	26	25	24	23
Kódovanie a testovanie	42	40	38	36
Integrácia a testovanie	16	19	22	25

Projekty s rozsahom do 2 KDL (do 2000 riadkov zdrojového textu) sa označujú ako malé, s rozsahom 2 až 8 KDL rozsiahlejšie, s rozsahom 8 až 32 stredne rozsiahle, 32 až 128 KDL rozsiahle.

Ďalšie dva typy projektov majú rozdelenie pracnosti do fáz mierne odlišné - vyšší je podiel vo fáze návrhu, integrácie a testovania na úkor kódovania. Odhady podielov pre iné rozsahy projektov je možné z tabuľky získať pomocou interpolácie.

6.7.1.2. Rozdelenie trvania projektu do fáz

Z odhadu pracnosti projektu nie je možné vypočítať odhadovaný čas riešenia projektu jednoduchým vydelením pracnosti počtom ľudí, ktorí sú pre riešenie k dispozícii. Takýto postup je možný len pri aktivitách, ktoré nemajú vysoké nároky na medziľudskú komunikáciu - tvorba softvéru k takýmto aktivitám nepatri. **Medzi dĺžkou trvania projektu a jeho pracnosťou je nelineárna závislosť**. Na zniženie doby trvania na polovicu nestačí zdvojnásobiť počet pracovníkov zúčastnených na projekte.

Platí tzv. **Brooksov zákon**:

"**Pridaním ľudskej sily do oneskoreného projektu, ho môžeme oneskoríť ešte viac.**"

Pre rozdelenie trvania projektu do jednotlivých fáz sa používa odhad percentuálneho podielu časov jednotlivých fáz:

Celkový odhadovaný rozsah projektu

Fáza	2 KDL	8 KDL	32 KDL	128 KDL
Návrh	19	19	19	19
Programovanie	63	59	55	51
Integrácia	18	22	26	30

6.7.2. Klasifikácia a popis plánovacích a kontrolných metód

Metódy projektového riadenia dosiahli v uplynulých rokoch značný rozvoj a aj praktické uplatnenie. Ich členenie nie je ešte ustálené. Možno ich rozdeliť, podľa rôznych hľadisk, napr. :

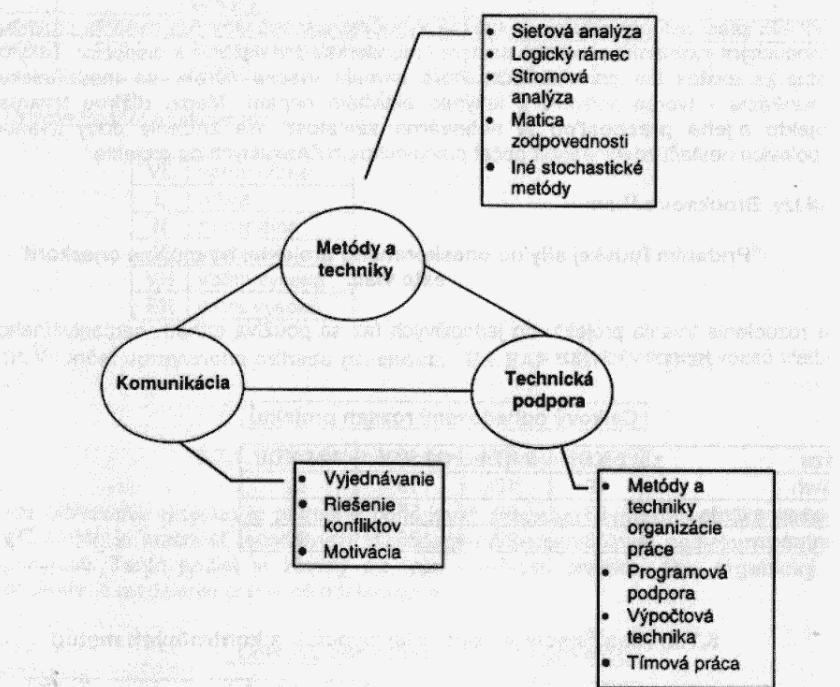
1. rozdelenie metód projektového riadenia podľa fázy uplatnenia :

- vo fáze stratégie
- taktiky
- zavedenia
- výhodnotenia a pod.

2. rozdelenie metód projektového riadenia podľa koncipovania projektov na :

- ideové
- obchodno-inžinierske
- technické

Realizácia jednotlivých metód je podmienená aj ďalšími oblastami, a to najmä **komunikáciou a technickou podporou** - tieto vzťahy znázorňuje Obr. 6.5.



Obr. 6.5 Klasifikácia metód projektového riadenia z hľadiska ich vzťahu ku komunikácii a technickej podpore

6.7.3. Metódy sietovej analýzy

Základom metód sietovej analýzy je grafické znázornenie pomocou sietového diagramu. Pod sietovým diagramom rozumieme konečný súvislý orientovaný acyklický graf, spravidla s jediným začiatocným a jediným koncovým uzlom. Pomocou jednoduchých prostriedkov - orientovaných úsečiek a uzlov - je možné popisať a graficky znázomiť každú prácu v ktorejkoľvek oblasti ľudskej činnosti. Význam grafickej reprezentácie rastie s počtom väzieb a závislostí medzi nimi.

Základné pravidlá zostavovania sietových diagramov:

1. Každá činnosť ma vždy jeden začiatocný a jeden koncový uzol. Začiatocný uzol sa označuje indexom i , koncový indexom j , takže každá činnosť je určená usporiadanou dvojicou (i,j) .
2. Sietový diagram má vždy jeden začiatocný a jeden koncový uzol, v prípade, že diagram má viac začiatocných príp. koncových uzlov, diagram sa normalizuje (t.j. vyberie sa jeden počiatocný (koncový) uzol a spojí sa s ostatnými fiktívnymi činnosťami).
3. Žiadna činnosť nemôže byť zahájena skôr, než sú dokončené všetky činnosti, ktoré jej bezprostredne predchádzajú.
4. Sietový diagram musí správne popisovať závislosť jednotlivých činností. Závislé činnosti sa oddeľujú od nezávislých pomocou fiktívnych činností. Závislosť činnosti sa vyskytuje všade tam, kde dve alebo viac činností končí alebo vychádza z jedného spoločného uzla.
5. Súbežné činnosti sa oddelujú v sietovom grafe fiktívnymi činnosťami. Cieľom týchto činností je znížiť stupeň neistoty dosiahnutia dielčieho cieľa. Znázornenie súbežných činností nesmie byť v rozpore s pravidlom o jednoznačnom zobrazení každej činnosti.
6. Fiktívne činnosti sa používajú:
 - k oddeleniu závislých a nezávislých činností
 - k oddeleniu súbežných činností
 - k vytvoreniu jedného počiatocného a jedného koncového uzla grafu
 Slúžia iba ako pomocný nástroj grafického znázornenia projektu. Nikdy nie sú spojené so spotrebou času ani s vynaložením nákladov, pretože reálne neexistujú.
7. Pri zobrazení niektorých akcií sa niekedy zaradujú do sietových diagramov tzv. predstihové činnosti (lead-time activity), ktoré sa predradujú pred počiatocný uzol. Predstihové činnosti vyjadrujú skutočnosť, že vlastné zahájenie celej akcie je viazané na skončenie niektorých prác, prípadne prijatie rozhodnutí, na čo riadiaci orgán zodpovedajúci za danú akciu nemá vplyv.
8. Agregácia činností. V praxi sa často vyskytujú prípady, keď je možné zahájiť niektorú z nasledujúcich činností ešte pred dokončením predchádzajúcich.
9. Rozsah a detailnosť diagramu závisí hlavne od toho, na akom stupni riadenia sa bude používať. zo tohto hľadiska rozoznávame tri stupne:
 - Prvý stupeň - sietový diagram sa uplatňuje vo vrcholovom riadení, prípadne pre širokú verejnosť. Počet činností nepresahuje 50 a nevyžaduje sa prísná logická nadváznosť.
 - Druhý stupeň - diagram je určený pre strednú úroveň riadenia. Jednotlivé činnosti ešte stále predstavujú veľké objemy prác, avšak už sa vyžaduje presná logická nadváznosť činností. Je to základ časovej a nákladovej kontroly akcie. Používajú sa jednotky: týždne, dekády, mesiace.
 - Tretí stupeň - používa sa pre operatívne riadenie danej akcie. Sú už podrobne členené, pretože musia umožniť riadenie a kontrolu malých skupín pracovníkov. Používajú sa časové jednotky deň, hodina.

Stupeň členenia sietového diagramu rovnako závisí na etape, v ktorej sa projekt nachádza. Pri definovaní činnosti je potrebné zaistiť rovnováhu medzi objemom vlastnej práce a objemom informácií o nej. Za vykonanie každej operácie vždy niekto zodpovedá.

- 10. Dĺžka úsečky**, ktorou znázorňujeme reálne, príp. fiktívne činnosti, nie je v žiadnom vzťahu k dobe trvania príslušnej činnosti.
- 11. Sieťový diagram nesmie obsahovať cyklus**. Vznik cyklu signalizuje logickú chybu (t.j. projekt nemôže byť v skutočnosti dokončený). V prípade, že dokážeme očíslovať všetky uzlové body diagramu tak, aby číselný index uzla na konci danej činnosti (j) bol vždy vyšší ako číselný index, z ktorého činnosť vychádza (i), potom je daný graf acyklický.

6.7.3.1. Metóda CPM

Metóda kritickej cesty (CPM - Critical Path Method) vychádza z práce autorov M. R. Walker a J. E. Kelly, ktorí v roku 1957 uverejnili metódu „Project Planning and Scheduling System“ (Systém plánovania a rozvrhovania projektov). Je to deterministický typ metódy a jej koncept tvorí základ sieťového plánovania. Metóde CPM vychádza z predpokladu, že existuje vždy aspoň jedna cesta pozostávajúca z následných činností s najdlhším trváním, ktorá determinuje dĺžku celého projektu. Táto cesta sa nazýva **kriticálna cesta**.

Rozbor procesu sa začína vymedzením **cieľa**, t.j. posledného časového uzla n . Od neho sa vraciame späť, definujeme činnosti, ktoré bezprostredne k nemu vedú, ich vstupné uzly atď. Základ prepočtu tvoria údaje o trvanií jednotlivých činností ($y_{i,j}$) - je to **normovaný čas trvania príslušnej činnosti**. Potom treba stanoviť čas začiatku činnosti (i, j). z pracovných postupov vyplýva, že činnosť (i, j) možno najskôr začať vtedy, keď sa dostaví časový uzol i . Tento čas označujeme $t_i(0)$ a platí:

$$t_j(0) = t_i(0) + y_{i,j}$$

T.j. čas konca činnosti (i, j) sa rovná času začiatku tejto činnosti zvýšenému o veľkosť normovaného času trvania tejto činnosti.

Najkratší čas, za ktorý je možné projekt skončiť, určíme ako maximálny súčet časov všetkých činností vytvárajúcich cestu v normalizovanom sieťovom grafe projektových činností od počiatocného uzla ku koncovému. Najskôr možné skončenie celého projektu môžeme porovnať s požadovaným časom skončenia projektu a podľa toho urobiť korekcie v projektových činnostiach alebo termínoch.

Pre činnosť (i, j) sú podstatné nasledujúce okamihy:

- najskôr možný začiatok
- najskôr možné skončenie
- najneskôr prípustný začiatok
- najneskôr prípustné skončenie

Ak najneskôr prípustný začiatok činnosti (j, k) a najskôr možné ukončenie predchádzajúcej činnosti (i, j), sú totožné, vtedy disponibilný čas sa rovná práve trvaniu činnosti (i, j) a táto činnosť je z hľadiska časového priebehu projektu **kriticálna činnosť**, pretože pripadné oneskorenie nemôže byť vyrovnané v rámci disponibilného času.

Ak sieť obsahuje aspoň jednu cestu zloženú z kritickej činností, táto cesta sa nazýva **kriticálna cesta**. Ak dôjde k oneskoreniu ľubovoľnej činnosti v kritickej ceste, dôjde aj k oneskoreniu celého projektu.

6.7.3.2. Metóda PERT

Metóda hodnotenia a previerky projektov (PERT - Project Evaluation and Review Techniques) bola vyuvinutá v roku 1958 nezávisle od seba pánnmi W. Fazar, J. Roseboom, C. Clark a D. Malcom pre potreby urýchlenia vývoja balistickej strely Polaris. Metóda PERT sa rozvinula pre potreby koordinácie a kontroly rozsiahlych projektov s veľkým počtom dodávateľov, kde trvanie prác je ťažké odhadnúť. Účelom je určiť pravdepodobnosť splnenia termínov a upozorniť na potrebu robiť opatrenia. Pri tejto najznámejšej stochastickej metóde sieťovej analýzy sa prvýkrát v operačnej analýze použili metódy znaleckého odhadu pre odhad trvania činnosti.

Pri metóde PERT sa odhaduje trvanie činností troma hodnotami:

1. **Optimistický odhad** trvania činnosti realizovanej pri príaznivých podmienkach
2. **Pesimistický odhad** trvania činnosti realizovanej pri nepriaznivých podmienkach
3. **Najpravdepodobnejšia doba trvania činnosti**.

Pri ďalších časových prepočtoch sa operuje iba s uvedenými hodnotami. Spôsob rozboru časového plánu sa zhoduje s metódou CPM.

V PERT diagramoch sa projekty reprezentujú pomocou termínov **udalosť** a **úloha**.

Udalosť (event) - niekedy sa nazýva aj mišník (milestone) - reprezentuje bod v čase významný z hľadiska začiatku alebo konca nejakej úlohy alebo skupiny úloh (činnosti alebo skupiny činností). Pre zobrazenie udalostí sa v PERT diagramoch používajú uzly (v rôznych implementáciách sú spravidla rôzne tvary týchto uzlov: kruh, obdĺžnik a pod.). V nasledujúcom príklade je použitý uzol v tvare elipsy. Každý uzol je rozdelený na tri sekcie, v ktorých je nasledujúce ohodnotenie uzla:

- **Identifikácia udalosti** - číselný kód
- **najskôr možný termín vzniku danej udalosti**
- **najneskôr prípustný termín vzniku danej udalosti**.

Namiesto reálnych dátumov a časov sa používa **relatívny čas** od začiatku projektu - t.j. začiatok projektu je v čase 0. Každý PERT diagram má práve jeden začiatok a koniec, ktorý zodpovedá udalosti - **začiatok projektu** a práve jeden koncový uzol, ktorý zodpovedá udalosti - **ukončenie projektu**.

Úloha (task) - niekedy sa nazýva aj aktivita (activity) sa v PERT diagrame označuje pomocou orientovanej hrany medzi dvomi udalosťami. Ohodnotenie hrany tvori:

- **Identifikácia úlohy** - alfabetický kód
- **očakávaná doba trvania úlohy** - predpokladaný čas potrebný pre vyriešenie úlohy (vypočítava sa z pesimistického, optimistického a najpravdepodobnejšieho odhadu času riešenia úlohy, vid. ďalej)

Orientácia hrany (i, j) od udalosti i k udalosti j vyjadruje skutočnosť, že udalosť i musí predchádzať udalosti j a začiatku riešenia úlohy (i, j) bezprostredne predchádza udalosť i a úloha končí udalosťou j .

Pre vyjadrenie závislosti medzi udalosťami sa používa orientovaná čiarkovaná hrana. Takáto hrana reprezentuje tzv. **fiktívnu (dummy) úlohu**, ktorej trvanie je nulové.

Odhad časových požiadaviek na projekt a odvodenie PERT diagramu:

Pred zostavením PERT diagramu je potrebné odhadnúť čas trvania každej projektovej úlohy. Diagram umožňuje indikovať odhadovaný najskôr možný čas, odhadovaný nejneskôr prípustný čas každej udalosti a očakávaný čas trvania každej úlohy v projekte. Hoci tieto časy sa často vyjadrujú v jednotkách človekoden (PD - person day), tento prístup sa neodporúča z nasledujúceho dôvodu: neexistuje lineárna závislosť medzi časom potrebným na realizáciu projektu a počtom ľudí v realizačnom tme:

Riešenie mnohých meškajúcich projektov sa ešte viac spomalilo zapojením ďalších ľudí.

Preto sa odporúča vyjadrovať čas v kalendárnych (alebo pracovných) dňoch potrebných na realizáciu úlohy daným počtom pracovníkov (priadených na túto úlohu). Nanešťastie neexistuje systém presných pravidiel pre odvodenie času trvania úloh - je potrebné používať kvalifikované odhady.

Predpokladajme, že sú známe časové požiadavky na realizáciu jednotlivých čiastkových úloh projektu. Postup zostavenia PERT diagramu bude nasledovný:

Zostavíme tabuľku všetkých úloh a udalostí projektu (PPT - Project Planning Table - tabuľka plánovania projektu) s nasledujúcimi stĺpcami:

ID úlohy (T)	Popis úlohy	ID udalosti (n)	Predchádzajúce udalosti (pre_n)	Nasledujúce udalosti (suc_n)	Očakávaný čas trvania (ED _T)	Najskôr možné ukončenie (ECT _n)	Najneskôr prípustné ukončenie (LCT _n)
--------------	-------------	-----------------	---------------------------------	------------------------------	--	---	---

Obsah a spôsob vyplnenia jednotlivých stĺpcov a riadkov PPT bude nasledujúci:

- Vytvorenie zoznamu všetkých udalostí a úloh v projekte - každej úlohe zodpovedá 1 riadok tabuľky.
- Vyplnenie nasledujúcich stĺpcov v PPT
 - ID úlohy - identifikátor úlohy
 - Popis úlohy
 - ID udalosti - identifikátor udalosti, ktorá predstavuje ukončenie danej úlohy
- Určenie závislostí medzi úlohami - vyplnenie stĺpcov:
 - Predchádzajúce udalosti - zoznam identifikátorov udalostí, ktoré musia bezprostredne predchádzať riešeniu danej úlohy
 - Nasledujúce udalosti - zoznam identifikátorov udalostí, ktoré musia bezprostredne nasledovať po ukončení danej úlohy
- Odhad trvania každej úlohy - vyplnenie stĺpcov:
 - Očakávaný čas trvania úlohy (ED - Expected Duration):

$$ED = (OT + 4 * MLT + PT) / 6$$

- OT (Optimistic Time) - je optimistický odhad trvania úlohy - je to čas, za ktorý je možné vykonať danú úlohu za predpokladu, že nenastanú

žiadne, ani bežné prerušenia práce (ako je napr. choroba zamestnanca)

- PT (Pessimistic Time) - je pesimistický odhad trvania úlohy, ktorý predpokladá, že "všetko, čo môže ísť zle, zle aj pôjde" (presne v duchu Murphyho zákonov)
- MLT (Most Likely Time) - je odhad najpravdepodobnejšej doby trvania úlohy. Tento čas nie je priemerom optimistického a pesimistického odhadu. Je to odhad, ktorý počíta s bežnými prerušeniami v práci.

5. Odvodenie času ukončenia každej úlohy - vyplnenie stĺpcov:

- Najskôr možné ukončenie - najskôr možný čas ukončenia úlohy (ECT - Earliest Completion Time) - t.j. najskôr možný termín vzniku udalosti zviazanej s koncom danej úlohy. Tento čas sa pre danú úlohu odvodí nasledovne:

Nech ukončenie úlohy predstavuje udalosť n. ECT_n pre udalosť n sa rovná najväčšiemu z časov ECT_{pre_n} (bzprostredne predchádzajúcich udalostí pre_n) zväčšených o odhadovaný čas trvania úlohy začajetej udalosťou pre_n a končiacej udalosťou n:

$$ECT_n = \max(ECT_{pre_n} + ED_{pre_n,n})$$

Pre prvú udalosť n=1 je najskôr čas ECT₁ = 0.

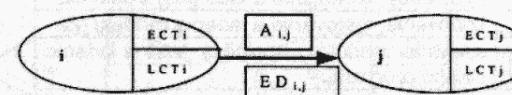
- Najneskôr prípustné ukončenie - najneskôr prípustný čas ukončenia úlohy (LCT - Latest Completion Time) - tento čas sa pre danú úlohu odvodí nasledovne:

Nech ukončenie úlohy predstavuje udalosť n. LCT_n pre udalosť n sa rovná najmenšiemu z časov bezprostredne nasledujúcich udalostí (suc_n) zmenšených o odhadovaný čas trvania úlohy, ktorá končí udalosťou n:

$$LCT_n = \min(LCT_{suc_n} - ED_{n,suc_n})$$

Pre poslednú udalosť m v projekte je najneskorší čas LCT_m = ECT_m.

6. Zostavanie PERT diagramu podľa identifikácie, následnosti a ohodnotení všetkých udalostí a úloh podľa tabuľky PPT a reprezentácie udalostí a úloh v projekte pomocou uzlov a hrán (nech úloha A_{i,j} končí udalosťou j a tejto úlohe bezprostredne predchádza udalosť i, t.j. úlohu A_{i,j} môžeme jednoznačne identifikovať aj usporiadanou dvojicou (i,j) resp. označením A_{i,j}:



Obr. 6.6 Reprezentácia a ohodnotenie uzlov a hrán v PERT diagrame.

Príklad. 6.1

V nasledujúcom zjednodušenom príklade plánovania čiastkového projektu (jednej fázy životného cyklu projektu - programovanie a ladenie) vychádzame z predpokladanej štrukturalizácie cieľového systému na 7 modulov M1, M2,...,M7. Kódovanie, testovanie a ladenie týchto modulov bude predstavovať dielčie úlohy. Nech poslednou úlohou v projekte bude spojenie a spoločné ladenie celej aplikácie. Dôležité udalosti sú

začiatok a koniec čiastkového projektu a ukončenie ladenia každého modulu zvlášť. Tieto udalosti budú identifikované celočíselnými identifikátormi 1,2,...9. Jednotlivé úlohy budú označené identifikátorom A, B, ...,H a pre jednoznačné priradenie k udalostiam aj usporiadanou dvojicou (i,j), kde i je udalosť, po ktorej bezprostredne môže úloha začať a j je udalosť, ktorou úloha končí (fiktívne úlohy nemajú identifikátor.)

1 ID úlohy (T _n)	2 Popis úlohy	3 ID udalosti (n)	4 Predchádzajúce udalosti (pre_n)	5 Nasledujúce udalosti (suc_n)	6 Očakávaný čas trvania (ED _n)	7 Najskôr možné ukončenie (ECT _n)	8 Najneskôr priпустné ukončenie (LCT _n)
Z (-1)	1 (začiatok projektu)	-	-	2	0	0	0 (3-3)
A (1,2)	2	1	-	3	3 (0+3)	3 (5-2)	-
B (2,3)	3	2	-	4	5 (3+2)	5 (7-2)	-
C (3,4)	4	3	2	5,6,7,8	2	7 (5+2)	(min(14-7, 13-6, 10-3, 14-2))
D (4,5)	5	4	3	8	7	14 (7+7)	14
E (4,6)	6	4	5	8	6	13 (7+6)	14
F (4,7)	7	4	6	8	3	10 (7+3)	14
(5,8)	fiktívna	8	5	9	0	14 (14+0)	14
(6,8)	fiktívna	8	6	9	0	13 (13+0)	14
(7,8)	fiktívna	8	7	9	0	10 (10+0)	14
G (4,8)		8	4,5,6,7	9	2	14 (max(7+2, 14+0,13+0,1 0+0))	14 (19-5)
H (8,9)	9 (ukončenie projektu)	8	-	-	5	19 (14+5)	19

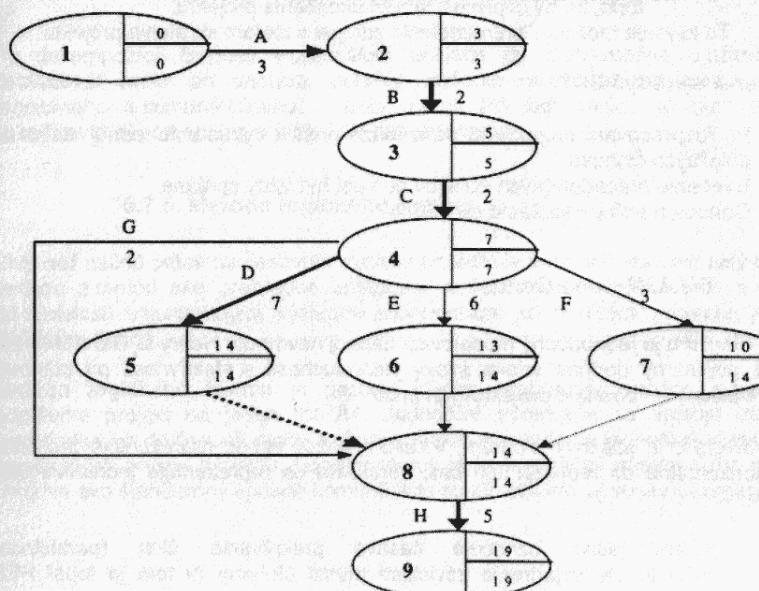
ID úlohy	Popis úlohy
A	kódovanie, testovanie a ladenie modulu M1
B	kódovanie, testovanie a ladenie modulu M2
C	kódovanie, testovanie a ladenie modulu M2
D	kódovanie, testovanie a ladenie modulu M2
E	kódovanie, testovanie a ladenie modulu M2
F	kódovanie, testovanie a ladenie modulu M2
G	kódovanie, testovanie a ladenie modulu M2
H	spojenie modulov, spoločný test a ladenie celého programu

Tab. 6.1 Tabuľka plánovania projektu pre Príklad. 6.1

V nasledujúcim detaile tabuľky z príkladu Príklad. 6.1 je pomocou šipiek naznačený spôsob výpočtu najskôr možného a najneskôr priпустného času ukončenia úloh.

1 T _n	2 n	3 pre_n	4 suc_n	5 ED _n	6 Najskôr možné ukončenie (ECT _n)	7 Najneskôr priпустné ukončenie (LCT _n)
B (2,3)	3	2	4	2	5 (3+2)	5 (7-2)
C (3,4)	4	3	5,6,7,8	2	7 (5+2)	(min(14-7, 13-6, 10-3, 14-2))
D (4,5)	5	4	8	7	14 (7+7)	14
E (4,6)	6	4	8	6	13 (7+6)	14
F (4,7)	7	4	8	3	10 (7+3)	14
(5,8)	8	5	9	0	14 (14+0)	14
(6,8)	8	6	9	0	13 (13+0)	14
(7,8)	8	7	9	0	10 (10+0)	14
G (4,8)		8	4,5,6,7	9	2 (max(7+2, 14+0,13+0,1 0+0))	14 (19-5)
H (8,9)	9 (ukončenie projektu)	8	-	5	19 (14+5)	19

- I najdenie predchádzajúcej udalosti pre_n k udalosti n=8
- II určenie najskôr možného času predchádzajúcej udalosti ETC_{pre_n}
- III použitie tohto času v súčte
- IV určenie očakávaného času trvania ED_{pre_n} a jeho použitie v súčte, najdenie maximálneho súčtu



Obr. 6.7 PERT diagram pre Príklad. 6.1.

Alternatívou metódou konštrukcie PERT diagramu je jeho odvodenie spätným plánovaním od udalosti odpovedajúcej ukončeniu projektu smerom dozadu k udalostiam a úloham, ktoré musia predchádzať.

Kritická cesta v PERT diagrame:

V diagrame z predošlého príkladu sú niektoré orientované hrany zvýraznené väčšou hrúbkou čiary (A B C D H). Sekvencia týchto hrán tvorí tzv. kritickú cestu.

Kritická cesta je postupnosť navzájom závislých projektových úloh, ktorá má najväčší súčet odhadovaných časov trvania. Je to cesta, v rámci ktorej neexistuje žiadna časová rezerva pre vykonanie úloh. Časová rezerva pre vykonanie úlohy je rozdiel medzi najskôr a najneskôr možným ukončením danej úlohy. Ak čas najskôr a najneskôr možného ukončenia nejakej úlohy je rovnaký, táto úloha leží na kritickej ceste a nazýva sa **kritická úloha**.

Najdôležitejšie vlastnosti metódy PERT sú:

- Postupy metódy nútia vedúceho organizovať a kvalifikovať dostupné informácie a určiť ďalšie potrebné informácie.
- PERT je stochastická metóda určená odhadom troch časov, na rozdiel od CPM, ktorá je deterministická (každá činnosť v CPM je určená normovaným časom y_i)
- Metóda umožňuje grafické znázornenie projektu a jeho najdôležitejších činností:
 - činnosti, ktoré musia byť starostlivo sledované, lebo môžu zapričíniť meškanie projektu
 - iných činností, ktoré majú nevyužitý čas a môžu sa omeškať bez toho, že by ovplyvnili termín ukončenia projektu.

To zvyšuje možnosť prerodzenia zdrojov s cieľom skratenia projektu.

Obmedzenia metódy PERT:

- Pri rozpracovaní projektovej siete môže prieť k vypusteniu jednej alebo viac dôležitých činností.
- Uvedenie precedenčných vzťahov nemusí byť vždy správne.
- Odhady terminov sú často skreslené.

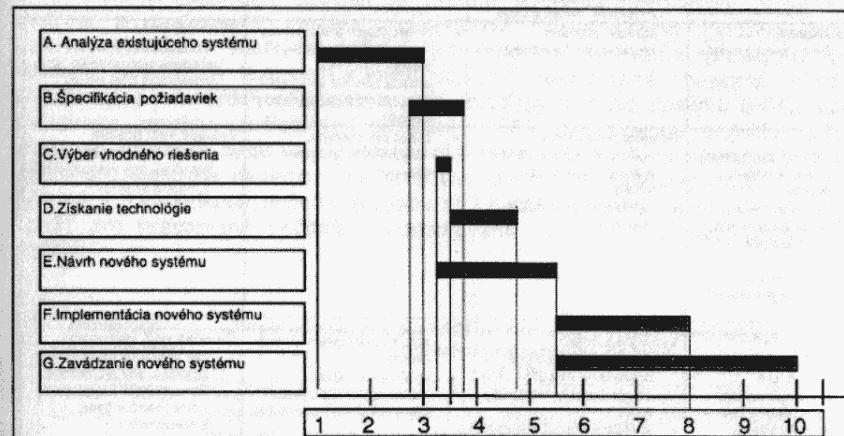
6.7.4. Metóda Gantt

Ganttov diagram je jednoduchý modelovací nástroj navrhnutý Henry L. Gantom v roku 1917. Je populárny dodnes vďaka svojej jednoduchosti a efektívnosti pri plánovaní projektu a sledovaní postupu projektových prác.

Ganttov diagram je stĺpcový diagram, v ktorom každý stĺpec reprezentuje projektovú úlohu. Horizontálna os reprezentuje čas, vertikálna os reprezentuje jednotlivé úlohy v projekte.

Ganttov diagram jasne označuje časové prekrývanie úloh (paralelnosť). Problematickejšie je ale vyjadrenie závislostí medzi úlohami (v tom je lepší PERT diagram).

Nasledujúci príklad ukazuje jednoduchý Ganttov diagram životného cyklu fiktívneho systému. Na vodorovnej časovej osi sú mesiace, počas ktorých sa predpokladá riešenie projektu. Na zvislej osi sú etapy riešenia projektu. V diagrame je možné uplatniť hierarchickú dekompozíciu činností až do potrebnej úrovne detailnosti.



Obr. 6.8 Príklad na použitie Ganttoveho diagramu

Po dekompozícii činností v predošлом príklade bolo možné a užitočné napr. detailizovať kroky pri analýze, detailné riešenie návrhu podsystémov, dátového, procesného a komunikačného modelu a pod. Pri jednotlivých úlohách je vhodné špecifikovať aj personálne a technické zabezpečenie.

6.7.5. Metóda logického rámcu

Skôr než začnú práce na časovom rozvahu projektu je potrebné stanoviť celý rad cieľov. Jednou z metód ako prehľadne zmapovať naše zámery a očakávania a uviesť ich do súvislosti s konkrétnymi výstupmi a činnosťami pri realizácii projektu, je metóda logického rámcu.

Metóda logického rámcu je postup, ktorým prehľadne, stručne a zrozumiteľne popíšeme projekt na jeden list A4. Jednotlivé informácie sa vpisujú do 16 polí, usporiadaných do štyroch radov a stĺpcov. Tako vyjadrujeme jednotlivé väzby a úroveň opisovaných prvkov projektu. Logický rámeček je komunikačný nástroj. Táto metóda sa používa ako štandardný spôsob komunikácie so zákazníkmi aj vo vnútri organizácie.

POPIS PROJEKTU	OBJEKTIVNE OVERITELNÉ UKAZOVATELE	PROSTREJDKY OVERENIA	PREDPOKLADY
Cieľ projektu • výšší cieľ projektu - zmena požadovaná na úrovni sektoru...	• podľa čoho poznáme, že sme prispeli k naplneniu výššieho cieľa • aj na tejto vysokej úrovni je potrebné stanovenie množstva, akosť a času	• aký zdroj údajov je pristupný alebo môže byť efektívne vytvorený z hľadiska nákladov	Cieľ voči vyššiemu cieľu • aké podmienky zaručujú, že splnený cieľ prispieje k cieľu na vysokej úrovni (mini log. rámec)
Účel projektu • dôvod vzniku projektu	• stav pri ukončení • ukazovatele úspešnosti	• zdroje údajov pre overenie ukazovateľov na tejto úrovni • je treba zahŕňať vytvorenie zdroja medzi výstupy alebo činnosti	Účel voči cieľu • ktoré vonkajšie predpoklady zaručujú, aby dosiahnutý účel mohol prispieť k splneniu cieľu?
Výstupy • to, čo má byť dodané, vytvorené v rámci projektu • projektový tím je priamo zodpovedný za dosiahnutie výstupu. • uvádzame 2 až 7 výstupov pre projekt	Dodacie podmienky • v skom množstve, čase a akosť treba dodať výstupy • vyzádzujeme funkčnosť výstupov	• zdroje pre overenie ukazovateľov na úrovni výstupov	Výstupy voči účelom • aké vonkajšie podmienky zaručujú, aby výstupy viedli k naplneniu účelu?
Cinnosti • hlavné zväzky činností pre dosiahnutie výstupu • 3-5 hl. skupín činností pre daný výstup • ľato hl. skupiny činností predstavujú štruktúru členenia práce	Vstupy a zdroje • čo všetko je treba pre prevedenie činnosti. Stručný prehľad materiálu, fudl... • je možné pripojiť informácie o zdrojoch (inštitúcie, fondy, partner...)	• zdroje údajov pre overenia činnosti • použitie rozpočtu, harmonogramy postupu, pracovné denníky, porady...	Činnosti voči výstupom • aké vonkajšie predpoklady zaručujú, aby prevedené činnosti viedli k výstupom v plánovanom čase a nákladoch?

Tab. 6.2 Tabuľková šablóna pre logický rámec

6.7.6. Iné stochastické metódy

Projektovací model „postupového vývoja systému“ je založený na myšlienke „pomaly rastúceho systému“. Neusiluje sa o absolútne riešenie s konečnou platnosťou, ale o približné ideálne riešenie orientované na skutočnosť.

Metóda koncepcie verzí

Mnohé problémy sa vyskytujú pri formulovaní cieľov hlavne preto, že nároky na rozvíjajúci sa systém nie je možné dopredu stanoviť (požiadavky na rozšírenie, zmeny v konkrétnom prostredí). Aby sa tomuto zabránilo, postupuje sa po navzájom nadvážujúcich krokoch. Na konci každého kroku je zlepšená verzia. *Dodržiavanie malých krokov prináša vyššiu pružnosť projektového riadenia* z hľadiska prispôsobovania.

Metóda konfiguračného riadenia

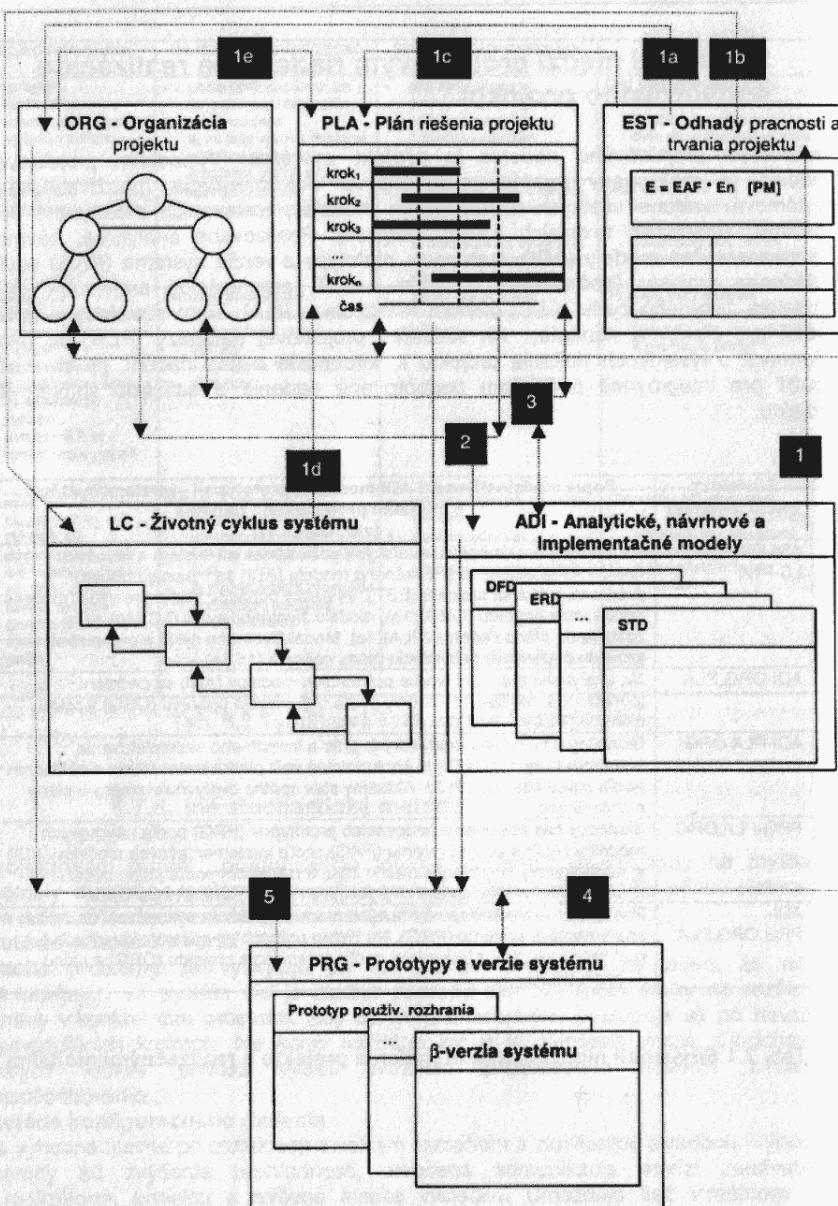
Je výhodná hlavne pri projektoch s veľkým rozsahom a zložitosťou skladbou. Výhodami metódy sú zvýšenie prehľadnosti, zlepšená komunikácia medzi zadávateľom a realizátormi projektu a zvýšená kvalita výsledku. Umožňuje tiež v reálnom čase registrovať zmeny vonkajšieho prostredia i požiadaviek, čím sa znížuje čas priebehu projektu. *Konfiguračné riadenie je metódou určovania, riadenia, sledovania a dokumentácie konfiguračných zmien počas priebehu projektu.*

7. Súvislosti medzi projektovým riadením a realizáciou softvérového projektu

Predmetom projektového riadenia je projekt systému. Výsledkom projektového riadenia je realizovaný systém a projektová dokumentácia (používateľská aj systémová, vrátane všetkých realizovaných modelov, testovacích prototypov, verzii systému, súvisiacich textových a iných popisov). Realizované analytické, návrhové a implementačné modely (ADI), testovacie prototypy a verzie systému (PRG) súvisia s riadením projektu (počiatocími odhadmi - EST, organizáciou projektu - ORG, modelom životného cyklu - LC, plánom realizácie - PLA). Tieto súvislosti je možné využiť pre efektívne riadenie. Ak súčasťou projektovej databázy (PDB) sú okrem informácií o výsledkoch riešenia projektu aj informácie o jeho riadení, je ich možné použiť pre integrované počítačom podporované riadenie a realizáciu softvérového projektu.

Č	Súvislosť medzi modelmi	Popis súvislostí medzi ADI modelmi, prototypmi, verziami systému a modelmi projektového riadenia (vid. Obr. 7.1)
	ADI-EST-ORG-LC-PLA	Na základe analýzy používateľských požiadaviek na systém a hubrého analytického dátového a funkčného modelu (ADI) sa vykonajú odhady pracnosti a trvania projektu (EST). Výsledky odhadov slúžia pre výber modelu organizácie projektu (ORG)(1a), modelu životného cyklu (LC)(1b) a pre zostavenie plánu riešenia (PLA)(1c). Model životného cyklu a organizácie projektu ovplyvňujú detailizáciu plánu riešenia (1d, 1e).
	ADI-ORG,PLA	Na analytické práce pri tvorbe potrebných modelov (ADI) sú pridelení jednotlivci a realizačné skupiny podľa organizácie projektu (ORG) a plánu riešenia (PLA).
	ADI-PLA,ORG	Skutočný čas začatia analytických prác a formálneho skompletovania potrebného modelu (ADI) je kontrolovaný voči plánovanému času a nákladom podľa plánu riešenia (PLA). Aktuálny stav späťne ovplyvňuje zmeny v pláne a organizácii.
	PRG-PLA,ORG	Skutočný čas zostavenia testovacích prototypov (PRG) podľa návrhových modelov (ADI) a verzí systému (PRG) podľa implementačných modelov (ADI) je kontrolovaný voči plánovanému času a nákladom podľa plánu riešenia (PLA). Aktuálny stav späťne ovplyvňuje zmeny v pláne a organizácii.
	ADI-PRG,ORG,PLA,	Podľa skompletovaných návrhových modelov (ADI) sa vykonávajú práce na implementačii systému (PRG). Na týchto práciach sa zúčastňujú jednotlivé realizačné skupiny a jednotlivci podľa organizácie projektu (ORG) a plánu riešenia (PLA).

Tab. 7.1 Súvislosti medzi modelmi riadenia projektu a realizačnými modelmi



Obr. 7.1 Súvislosti medzi modelmi riadenia a realizácie projektu

8. Nástroje, metódy a metodológie

Metodológie integrujú použitie **nástrojov** a **metód** použitia týchto nástrojov analýzy, návrhu a implementácie pre viaceru fáz životného cyklu a detailizujú toto použitie v presne špecifikovaných **krokoch** a **podmienkach** ich vykonania. Podmienky sú založené na:

- formálnych testoch existujúcich súvislostí modelov podľa pravidiel konzistencie, na testoch úplnosti a syntaktickej správnosti modelov
- neformálnych testoch splnenia používateľských požiadaviek na systém.

Oproti klasickým metodológiám štruktúrovanej analýzy a návrhu sa v súčasnosti dostávajú do popredia objektové metodológie.

Základom všetkých **štruktúrovaných metodológií** je funkčná a dátová štrukturalizácia systému pomocou nástrojov dátového a funkčného modelovania. Medzi najčastejšie v súčasnosti používané metodológie patria: YSM (Yourdon Structured Method), SSADM (System Structured Analysis/Design Method), Oracle*Method. Líšia sa hlavne úrovňou prepracovanosti, detailnosti krokov, doporučovaným postupom štrukturalizácie, dôrazom na funkčný alebo dátový pohľad a modifikáciami použitých modelovacích nástrojov. Základným nástrojom pre dátové modelovanie všetkých týchto metodológií je entitno-relačný diagram (ERD), základom funkčného modelovania je diagram dátových tokov (DFD).

Objektové metodológie sa zakladajú na integrovaní dátového a funkčného pohľadu do prirodenejšího modelu na báze tried a ich inštancií - objektov. Základným statickým pohľadom na systém je model na báze diagramu tried, základný dynamicky pohľad je na báze stavových diagramov. Jednotlivé metodológie sa líšia v použití ďalších nástrojov. Perspektívna je unifikácia objektových metodológií v podobe UML (Unified Modeling Language). Vychádza z objektových metodológií OMT (J. Rumbaugh), metodológie G. Boocha a Use-Case Modelu I. Jacobsona.

Výhoda objektového prístupu k analýze a návrhu oproti štruktúrovanému je najmä v nasledujúcich prednostiach:

- objekty sú bližšie používateľskému pohľadu na systém aj programátorskému pohľadu na jeho relačáciu – uľahčuje to komunikácie medzi analytikmi a používateľmi a analytikmi a programátormi
- objekty sú jediným hlavným modelovacím nástrojom (na rozdiel od dátových entít a funkcií pri štruktúrovanom prístupe) – prechod od prehľadových k detailným a od logických k fyzickým modelom sa vykonáva v podstate detailizačou a dekompozíciou jediného modelu – modelu tried.

9. CASE nástroje, systémy a technológie

Skratka **CASE** sa stala symbolom doby vo svete ľudí, ktorí majú niečo spoločné s vývojom rozsiahlych programových systémov. Pôvodný neskrátený názov "Computer Aided Software Engineering" znie menej tajuplnie obchodnícky a vyjadruje podstatu: využitie počítačov pre inžinierske činnosti spojené s analýzou, vývojom a údržbou

počítačových programov. Za týmto názvom sa neskrýva nič nové - pri malom zjednodušení môžeme povedať, že prvé kroky v počítačom podporovanom softvérovom inžinierstve vykonal človek už v dobe počítačového praveku (nie až tak veľmi vzdialenej), keď zistil, že pre prípravu programov sa dajú použiť programy a nielen tlačidlá ovládacieho panelu počítača...

Hlavne z komerčného hľadiska dnes málokto ku CASE systémom alebo nástrojom zaraďuje komplikátor, ladiaci prostriedok alebo pomocný program *make*, aj keď kontinuita medzi týmito nástrojmi a dnešnými CASE nástrojmi a systémami existuje.

Programové prostriedky, ktoré slúžia pre automatizáciu niektorého kroku alebo etapy životného cyklu označujeme ako **vývojové prostriedky**. V nasledujúcej tabuľke je prehľad vývojových prostriedkov a ich zodpovedajúcej formálnej reprezentácie využívanej systému.

Vývojový prostriedok	Použitá formálna reprezentácia využívanejho systému
assembleri komplikátory interpreti ladiace prostriedky syntaxou riadené editory integrované vývojové prostredia s 3GL a 4GL programovacími jazykmi nástroje pre údržbu verzii aplikácie generátory kódu z textových špecifikácií (iných ako programovacie jazyky) generátory pre prototypovanie diaľkových systémov (generátory obrazoviek, menu) systémy pre podporu štrukturovaného programovania CASE nástroje CASE systémy integrované CASE systémy	TEXTOVÉ ŠPECIFIKÁCIE
	GRAFICKÉ ŠPECIFIKÁCIE

Všetky uvedené prostriedky majú jeden spoločný cieľ: využiť v procese vývoja a údržby programových systémov **programovo implementované formálne abstrakcie**, pokiaľ možno čo najblížie predstavám ľudi (spočiatku tvorcov, dnes už konečne aj používateľov programových systémov). V ideálnom prípade by na začiatku tohto procesu mali byť **predstavy o požadovanom systéme a na konci vykonateľný kód** tohto systému. Pri použíti komplikátorov a interpretov je zatiaľ potrebné predstavy o systéme **formalizovať** pracne pomocou programovacieho jazyka (textovo). CASE nástroje a systémy sú programové prostriedky, ktoré umožňujú tieto predstavy zobrazovať graficky tabuľkovo alebo textovo pomocou rozličných modelovacích nástrojov – diagramov, tabuľiek (matic), popisov ponocou formálnych špecifikačných jazykov.

CASE nástroje poskytujú modelovacie prostriedky obyčajne pre jednu fazu životného cyklu alebo pre jeden pohľad na systém (napr. pre dátové modelovanie).

CASE systémy poskytujú viac rozličných modelov pre niekoľko fáz životného cyklu. CASE systémy sa spravidla klasifikujú z hľadiska fáz životného cyklu, ktoré podporujú, na 3 úrovne:

- **Upper CASE** - podporujú plánovanie a riadenie projektu. Poskytujú modelovacie a vyhodnocovacie prostriedky pre plán riešenia jednotlivých úloh spojených s projektom programového systému, pre odhad nákladov a času riešenia.
- **Middle CASE** - poskytujú prostriedky pre analýzu a návrh programových systémov (tzv. programovanie vo "veľkom")
- **Lower CASE** - poskytujú prostriedky pre podporu implementácie programových systémov alebo ich časti - generovanie prototypov, šablón, obrazoviek a pod. (tzv. programovanie v "malom").

Systémy s integrovanými všetkými troma úrovňami sa nazývajú **integrované CASE systémy**.

Softvérová technológia je spojenie metodológie a použitých vývojových softvérových prostriedkov. Pri použíti CASE systémov sa používa pojmenovanie CASE technológie.

10. Záver a pohľad do budúcnosti

Pre vývoj rozsiahlych a zložitých systémov sú perspektívne integrované CASE systémy – umožňujú počítačom podporované riadenie a realizáciu softvérových projektov.

Základné požiadavky na integrovaný CASE systém (InCASE) sú:

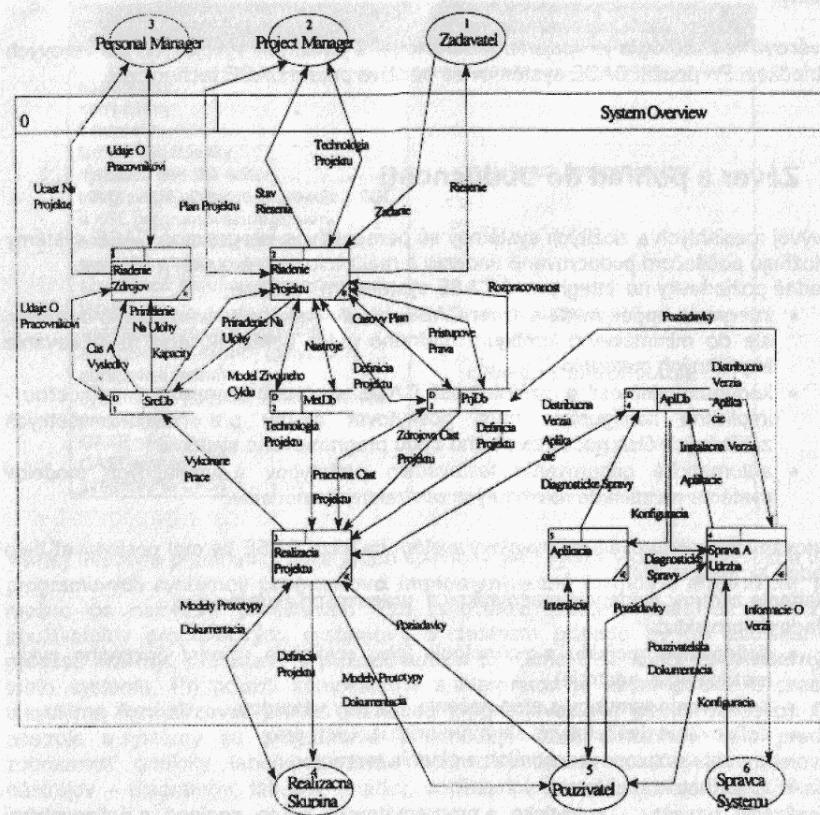
- integrácia upper, middle a lower CASE, nie do veľkého univerzálneho systému, ale do minimálneho konfigurovateľného jadra umožňujúceho modifikovanie implicitných metodológií
- konfigurovatelnosť a rozširiteľnosť CASE systému nesmie byť nutnosťou - implicitná konfigurácia musí poskytovať služby pre riešenie všetkých základných úloh počas životného cyklu programového systému
- automatické generovanie testovacích prototypov a simulačných modelov systému na základe navrhnutých abstraktných modelov.

Integrovaný rekonfigurovateľný vývojový systém na báze CASE by mal poskytovať tieto základné skupiny služieb:

- Riadenie zdrojov: evidencia pracovníkov a pracovných skupín
- Riadenie projektu:
 - definovanie projektu a technológie jeho realizácie (model životného cyklu, metodológia, nástroje)
 - definovanie termínov a etáp riešenia, odhady nákladov
 - definovanie riešiteľských skupín a prístupových práv
 - kontrola rozpracovanosti úloh a plnenia termínov
 - vyhodnotenie realizovaných prác
- Realizácia projektu - analytické a programátorské práce spojené s definovanými fázami životného cyklu, zostavovaním potrebných modelov a prototypov
 - evidovanie používateľských požiadaviek
 - zostavenie potrebných modelov
 - testy konzistencie modelov

- evidovanie splnenia používateľských požiadaviek nejakou časťou projektu
 - generovanie potrebných prototypov
 - generovanie simulačných modelov
 - generovanie potrebných zdrojových textov
 - Správa, údržba a prevádzka aplikácie
 - evidencia chybových stavov
 - sledovanie využívania služieb a časových odoziev systému
 - evidencia nových požiadaviek a prípmienok používateľov
 - správa verzií systému
 - inštalácia a konfigurovanie systému alebo jeho častí do skúšobnej a ostrej prevádzky

Nasledujúci diagram dátových tokov popisuje uvedené základné skupiny služieb vo vzťahu k subjektom, pre ktoré sú služby určené.



Obr. 10.1 Systémový diagram služieb integrovaného CASE systému

Pretože integrovaný CASE má poskytnúť potrebné informácie pre všetky etapy životného cyklu programov, je potrebné evidovať údaje o programovom systéme z troch hľadiší: z pohľadu riadenia projektových prác, z pohľadu realizácie projektu a z pohľadu používania programového systému. Takéto štruktúrovanie viedie k nasledujúcim základným entitám dátového modelu InCASE:

STRUKTÚRA PROJEKTU PROGRAM. SYSTÉMU	ANALYTICKÝ POHĽAD NA PROGRAMOVÝ SYSTÉM	POUŽIVATEĽSKÝ POHĽAD NA PROGRAMOVÝ SYSTÉM
projekt systému	systém	systém
projekt podsystému	podsystém	podsystém
fáza projektu	proces	služba
pohľad (DM,FM,CM)	modul	dialógový riadok, okno, obrazovka
model (ERD,DFD,...)	objekt	dialogový prvok
príkaz modelu	funkcia	
atribút príkazu	udalosť/reakcia	

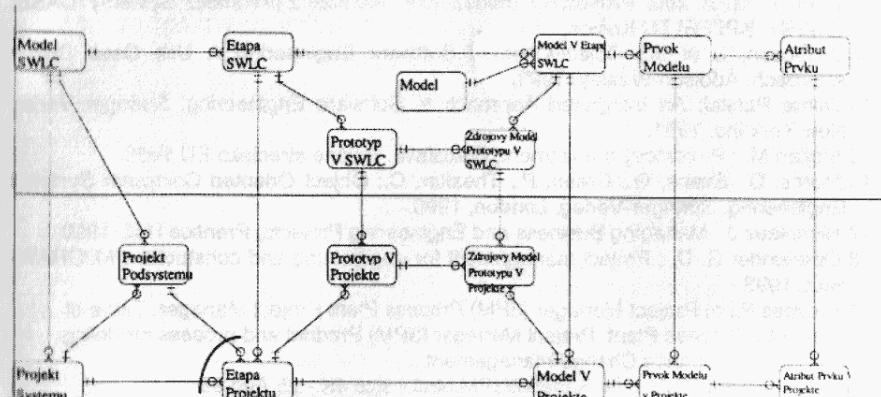
Tab. 10.1 Štruktúrovanie programového systému z hľadiska projektu - analýzy/syntézy - používania

Dátový model je možné rozdeliť do štyroch častí

SrcDb	databáze zdrojov - pracovníci a využitie ich kapacity na projekte
MetDb	databáza pre metodológiu a nástroje používanej technológie
PrjDb	projektová databáza definovanie projektu, časový plán úloh, definovanie prístupových práv k časťiam projektu, výsledky jednotlivých fáz (analýza, architektúra, návrh, implementácia), časti projektu a ich rozpracovanosť, požiadavky na zmeny a nové služby
ApplDb	databáza aplikácia (zdrojové texty, uživatelská súťaž, dokumentácia)

Pre konfigurovatelnosť technológie v integrovanom CASE je potrebné zabezpečiť prepojenie modelu životného cyklu a v ňom používaných nástrojov (MetDb) s projektovou databázou (PriDb) napr. podľa nasledujúceho modelu:

Definícia technológie



Obr. 10.2 Jadro logického dátového modelu integrovaného CASE systému

11. Použité a odporúčané zdroje informácií

1. Alexy J. : Systémová a operačná analýza v riadení podnikov, ALFA 1985.
2. Analysis Stage Checklist: Project Manager Analysis Stage Checklist Role : Project Manager. Definition of Role: The Project Manager role includes representatives from both ORACLE and Georgia Tech...
http://danshiki.oit.gatech.edu/html/documents/tag/Analysis_R_Proj_Mgr.html - size 3K - 13 Dec 94
3. Analysis Stage Checklist: Project Manager Analysis Stage Checklist Role : Project Manager. Definition of Role: The Project Manager role includes representatives from both ORACLE and Georgia Tech...
http://zappa.oit.gatech.edu/html/documents/tag/Analysis_R_Proj_Mgr.html - size 3K - 10 Mar 95
4. Boehm Barry,W.: Software Engineering Economics. IEEE Transactions on Software Engineering, Vol.SE-10, No.1, January 1984.
5. Boehm Barry,W.: A Spiral Model of Software Development and Enhancement. IEEE Computer, May 1988 pp.61-72.
6. Brandejský T. : CA SuperProject, BAJT 3/1992.
7. Contract for Application Outsourcing Project. BOSTON, MA --February 10, 1995 -- Keane, ... <http://www.keane.com/whatsnew/eana.html> - size 3K - 1 Apr 96
8. De Marco, T.: Structured Analysis and System Specification, Englewood Cliffs, Prentice Hall, 1978.
9. Havlice, Z.: Conception of integrated CASE system. Proceedings of Scientific Conference with International Participation - Electronic Computers & Informatics. September 1996, Košice - Herľany, pp.254-259.
10. Havlice, Z.: Prototypovanie a modelovanie rozsiahlych informačných systémov. In: Proceedings of FEI'95 CONFERENCE on Electronic Computers and Informatics, Sept. 22-23, 1994 Košice-Herľany s.237-242.
11. <http://www.cs.bris.ac.uk/~wong/Proj/main.htm> - Project management with Fuzzy Logic
12. Choma, M. a kol.: Projektový manažment. Referát z predmetu Systémy CASE, 1996/97, KPI FEI TU Košice.
13. Jacobson, I. et al.: Object-Oriented Software Engineering A Use Case Driven Approach. Addison-Wesley, 1993.
14. Jalote Pankaj: An Integrated Approach to Software Engineering. Springer Verlag New York Inc. 1991.
15. Majtán M. : Projektový manažment, Bratislava, Edičné stredisko EU 1995.
16. Morris, D., Evans, G., Green, P., Theaker, C.: Object Oriented Computer Systems Engineering. Springer-Verlag, London, 1996.
17. Nicholaus J.: Managing Business and Engineering Projects, Prentice Hall, 1990
18. Oberleider G. D. : Project management for engineering and construction, McGRAW-HILL 1993
19. Process Plant Project Manager (3PM) Process Plant Project Manager. Table of Contents. Process Plant Project Manager (3PM) Product and process modeling History maintenance Change management...
<http://www.cs.hut.fi/~musyk/leaflet/3PM.html> - size 4K - 25 Aug 95
20. Project Manager (PjM) Next: Software Engineer (SE).
<http://www.umcs.maine.edu/~ftp/wisr/wisr7/dpewg/node7.html> - size 2K - 19 Sep 95

21. Project Manager Today Project Manager Today ---- May 1996. ISEB Diploma launched. The Project Management Certificate for the IT industry was launched by the ISEB, in 1990... <http://www.projectnet.co.uk/pm/pmt/pmtmayne.htm> - size 5K - 16
22. Project Manager's Palette-PMtalk PMtalk -- A Discussion Forum for Project Managers. A moderated forum dedicated to the discussion of cross-functional corporate project management. Topics..<http://www.4pm.com/discussion.html> - size 2K - 21 Jun 96
23. Rambaugh, J. , Blaha M., Premerlani W., Eddy F., Lorenzen W.: Object-Oriented Modeling and Design, Prentice-Hall 1991.
24. Richta K.: Projektování programových systémů. ČVUT Praha, 1994.
25. Role of the Project Manager Role of the Project Manager. The Project Manager is appointed by Facilities Management as its representative to the customer. The same individual will act. <http://www.abs.uci.edu/depts/facil/renovate/projmang.html> - size 2K - 20 Jun 96
26. Russev,S., Adamec,M., Brdiar,J.: Softvérové inžinierstvo a systémy CASE, Ekonomická Univerzita Bratislava, 1993.
27. Šešera L., Mičovský A.: Objektovo-orientovaná tvorba systémov a jazyk C++. PERFEKT Bratislava, 1994.
28. Whitten, J. L., Bentley L. D., Barlow V. M.: Systems Analysis & Design Methods. Second edition. IRWIN 1989, Boston.
29. Yahdav D. : Sledování nepolapitelného projektu ,BAJT 8/1992.
30. Yourdon E.: Modern Structured Analysis, Prentice-Hall, 1989.

12. Zoznam skratiek a značiek

SKRATKA	VÝZNAM	POUŽITIE
A	operácia archivácie	43
ADI	analytické, návrhové a implementačné modely	75
AUT	abstraktívý údajový typ	25
BNF	Backus-Naur form – metajazyk pre popis syntaxe	19
CASE	Computer Aides Software Engineering – počítačom podporované softvérové inžinierstvo	11, 22, 30, 34, 42, 45, 77, 79
CD	Class Diagram – diagram tried	25, 35, 49
CM	Control Model – riadiaci model	8, 16, 17, 44
CoCoMo	Constructive Cost Model	60
CPM	Critical Path Method – metóda kritickej cesty	66
D	Delete – operácia zrušenia	41
DF	Dataflow – dátový tok	28
DFD	Data Flow Diagram – diagram dátových tokov	4, 27, 28, 35, 45, 77
DI	doménová integrita	41
DK	kaskádne zrušenie	42
DM	Data Model – dátový model	8, 16, 25, 44
DN	nulitné zrušenie	42
DR	reštrikčné zrušenie	42
DS	Datastore – dátová pamäť	28
DSD	Data Structure Diagram – diagram dátových štruktúr	18, 45
EAF	Effort Adjustment Faktor – upravujúci faktor úsilia	61
ECT	Earliest Completion Time – najskôr možné ukončenie	69
ED	Expected Duration – očakávaný čas trvania	68
ELH	Entity Life History – životný cyklus entít	40
ELM	Event List Matrix – matica udalostí	31
En	Nominal Effort – nominálny odhad pracnosti	60
ERD	Entity-Relationship Diagram – entitno-relačný diagram	20, 42, 45, 77
ER-model	entitno-relačný model	9
EST	počítačné odhady	75
FHD	Function Hierarchy Diagram – diagram hierarchie funkcií	17
FK	Foreign Key – cudzí kľúč	43
FM	Function Model – funkčný (procesný) model	8, 16, 17, 25, 44
FSD	Forms Sequence Diagram – diagram postupnosti obrazoviek	27, 39, 46
I	Insert – operácia vloženia	41
ID	identifikátor	68
IK	kaskádne vloženie	42
IN	nulitné vloženie	42
IR	reštrikčné vloženie	42
JSD	Jackson Structure Diagram – Jacksonov diagram štruktúry	18, 27, 32
JSP	Jackson Structured Programming – Jacksonova metóda štruktúrovaného programovania	18
K	kaskádny	41
KDL	Kilo Delivered Lines – tisíc riadkov zdrojového textu	60
KSA	konečno-stavový automat	35
LC	Life Cycle – životný cyklus	75
LCT	Latest Completion Time – najneskôr prípustné ukončenie	69
N	nulitný	41
OM	Object Model – objektový model	25, 49
OMT	Object Modeling Technique – Rumbaughova objektovo-orientovaná metodológia analýzy a návrhu	11, 77

OO/A/D	Object-Oriented Analysis/Design – objektovo-orientovaná analýza/návrh	25
OOP	Object-Oriented Programming – objektovo orientované programovanie	3
ORG	organizácia projektu	75
PD	Person-Day – človekoden	68
PDB	Project Data Base – projektová databáza	75
PERT	Project Evaluation and Review Techniques – metóda hodnotenia a previerky projektov	40, 67
PK	Primary Key – primárny kľúč	43
PLA	plán realizácie	75
PM	Process Model – procesný (funkčný) model	16
PM	Person-Month – človekomesiac	60
PPT	Project Planning Table – tabuľka plánovania projektu	68
PR	používateľské rozhranie	8
PRG	testovacie prototypy a verzie systému	75
R	reštrikčný	41
R	Read – operácia čítania	43
RI	referenčná integrita	41
SCD	Structure Chart Diagram – diagram modulárnej štruktúry	27, 32, 46
SD	Scenario Diagram – diagram scenárov	27, 37, 49
SP	starší programátor	58
SSADM	System Structured Analysis/Design Method – britská metodológia štruktúrovanej analýzy a návrhu	20, 77
STD	State Transition Diagram - stavový diagram	35, 39, 46, 49
SwLC	Software Life Cycle – životný cyklus programov	10
U	Update – operácia aktualizácie	41
UCD	Use Case Diagram – diagram činností	31, 49
UIM	User Interface Model – model používateľského rozhrania	16, 39
UK	kaskádna aktualizácia	42
UML	Unified Modeling Language – unifikovaný modelovací jazyk	25, 77
UN	nulitná aktualizácia	42
UR	reštrikčná aktualizácia	42
VDM	Vienna Development Method – formálny špecifikačný jazyk	4
VP	vedúci programátor	58
YSM	Yourdon Structured Method – Yourdonova štruktúrováná metodológia	11, 77

