1b. Strukturovaná analýza. Objektová analýza a návrh, UML. Nástroje a modely datové, funkční a časové dimenze systému. Softwarové metriky. CMM. Odhady COCOMO a funkční body.

Osnova

1. Strukturovaná analýza
2. Objektová analýza a návrh, UML
3. Nástroje a modely datové, funkční a časové dimenze systému
4. Softwarové metriky
5. CMM
6. Odhady COCOMO a funkční body

Výklad

1. Strukturovaná analýza

Strukturované metody:

- Člení projekt na malé, dobře definované aktivity a určují posloupnost a interakci těchto aktivit
- Efektivněji využívá zkušené i méně zkušené pracovníky

Rozdělení projektu na jednotlivé etapy a dílčí kroky umožňuje lepší odhad času, kontrolu výsledků a včasné řídící zásahy.

Při strukturované analýze je systém zkoumán zejména ze dvou základních pohledů (postup při dekompozici systému):

- **funkčně orientovaný přístup** – Systém je chápán jako množina interagujících funkcí. Funkční transformace jsou umístěny v procesech, které jsou propojeny datovými a řídícími toky.
- **datově orientovaný přístup** – Hledá fundamentální datové struktury aplikace. Funkční stránka (tj. různé transformace dat) je méně podstatná. Datový model definuje konceptuální model pro DB systému.
- SA oba dva přístupy vzájemně kombinuje – oddělené funkční a datové modely.
- Postupné zpřesňování modelů.
- Snaha o zachování konzistence: uvnitř modelu a vzájemně mezi jednotlivými modely.

YMSA

- E.Yourdon – Moderní strukturovaná analýza (1989)
- reakce na kritiku Analýzy pomocí 4 modelů (viz níže)
    - Při formulaci fyzického modelu ví uživatel o systému více, než analytik. Má pocit, že „analytik tomu nerozumí, za moje peníze se to učí“.
    - Uživatel odmítá spolupráci na vývoji logického modelu. „Pokud analytik neuměl stvořit sám ani fyzický model, tak nemůže navrhnout dobře ani nový systém.“

    - Analytická fáze je „období nic nedělání“, skutečná práce začne, až když se začne programovat. Tvorba 4 modelů prodlužuje období „nic nedělání“.
    - Mnoho uživatelů navíc zpochybňuje smysl podrobné a pečlivé tvorby modelu systému, který bude stejně! zahozen a nahrazen novým systémem.
- analýza s esenciálním modelem (dlouhodobě stabilní systém)
- z esenciálního modelu se odvodí implementační model

Esenciální model

Skládá se ze dvou částí:

1) Model okolí

- popisuje rozhraní mezi systémem a okolním světem
- kontextový diagram (DFD s jediným procesom + terminátory – lidé a systémy z okolí, s nimiž systém komunikuje)
- seznam událostí: toky (flow), časové události (temporal), řídící události (control)
- prvotní dátový slovník (datová rozhraní mezi systémem a terminátory)

2) Model chování systému

- popisuje vnitřní části systému
- dekompozici systému na jednotlivé procesy, toky a paměti uspořádané v sadě DFD a doplněné o příslušné minispecifikace a definici datových komponent
- uplatňuje sa postup zdola-nahoru
- prvotní model chování: prvotní systémový DFD + ERD a datový slovník
- tvorba esenciálního modelu: vyvažování prvotního modelu chování (směrem nahoru – seskupování, i směrem dolů – dekompozice)
- kontrola konzistentnosti DFD s ERD
- definování procesů pomocí minispecifikací a řídících procesů pomocí STD (stavových diagramů)

Implementační model

- vybere se jenom automatizovaná část systému, manuální část se převede na terminátory
- určení uživatelského rozhraní
- doporučení pro návrh rozhraní
- návrh formulářů
- identifikace manuálních podpůrných aktivit
- opatření pro chybové technologie
- specifikace operačních omezení

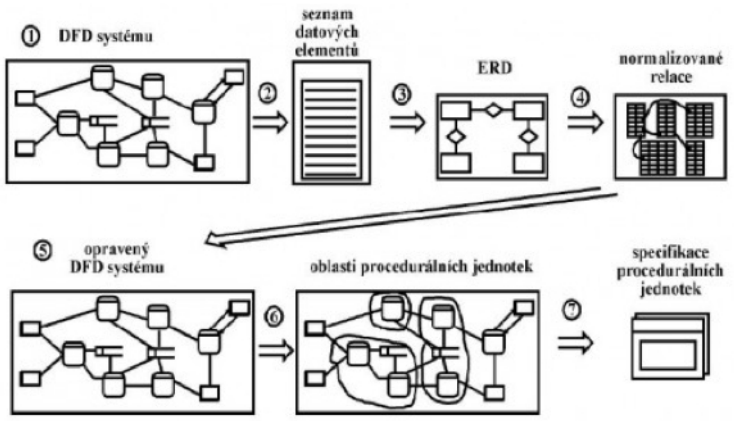
Postup:

1. vytvoření modelu okolí systému
2. vytvoření prvotního modelu chování systému
3. dokončení esenciálního modelu
4. vytvoření implementačního modelu

Další metody:

- SASS
  - Structured Analysis and System Specification
  - DeMarco (1978) – připisováno prvenství při použití DFD
  - shora-dolů
  - Analýza pomocí 4 modelů
  - Základní kroky metodiky
    1. Studie stávajícího fyzického systému
    2. Odvození logického ekvivalentu stávajícího systému
    3. Odvození nového logického systému
    4. Odvození nového fyzického systému
    5. Kvantifikace cen a termínů
    6. Výběr jedné z možností
    7. Začlenění nového fyzického modelu do specifikace

- Logické modelování
  - Gane, Sarson (1979) – DFD + ERD



- Pohledová analýza
  - View Point Analysis - VPA
  - Mullery (1979)
  - zdola-nahoru

- Použití
  - hierarchie entit není dosud vytvořena,
  - hierarchie entit není na první pohled zřejmá,
  - systém není přirozeně hierarchicky uspořádán
- Kroky
  1. identifikace pozorovacích bodů
  2. sdružení pohledů do skupin (funkční, nefunkční, hraniční, definiční pohledy)
  3. určení struktury pohledů

- DSSD
  - Data Structure Systems Development
  - datově orientovaný přístup
  - nejedná se o striktně stanovenou metodiku, spíše jde o souhrn zkušeností, které vedly ve většině případů k úspěchu

- SSADM
  - Structured System Analysis and Design Methodology
  - důraz kladen na detailní a strukturovaný přístup v každé etapě životního cyklu vývoje
  - používá se při vývoji systémů, ale nepokrývá celý životní cyklus systému
  - 3 základní pohledy na informační systém
    - Logické datové struktury - LDS
    - Diagramy datových toků - DFD
    - Životní cykly entit - ELH (Entity Life History)
  - Etapy
    1. stávající systém je analyzován s cílem porozumět problémové oblasti nového systému.
    2. pohled na stávající systém je použit pro vytvoření specifikace požadavků systému.
    3. specifikace požadavků je zpracována detailně, takže je možné formulovat podrobné technické možnosti a alternativy.
    4. současně s následujícím krokem (návrh log. procesů) je vypracován návrh logických dat. Vznikající modely jsou navzájem neustále vyvažovány.
    5. je vytvořen návrh logických procesů (viz. předchozí etapa).
    6. logický návrh je převeden na fyzický návrh při aplikaci jednoduchých pravidel.

2. Objektová analýza a návrh, UML

Objektová analýza a návrh

- objekty kombinují data a funkce do uzavřené podoby, soudržné jednotky
- objekty ukrývají data za vrstvou funkcí (operací)
- modeluje požadavky na systém prostřednictvím objektů a jimi poskytovaných metod
- objekty se oproti funkcím (příliš) nemění, tedy OO model tolik nezestárne a proto je lépe udržovatelný

Proč objektově orientovaná analýza

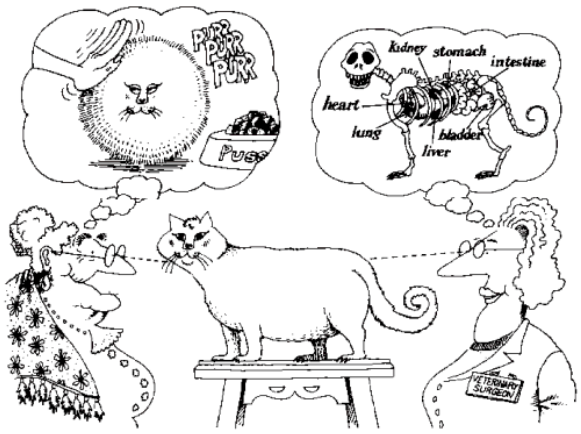
- Zvládnutí náročnějších problémových oblastí
- Zlepšení interakce mezi analytikem a expertem problémové oblasti
- Zvýšení vnitřní konzistence analytických výsledků
- Explicitní vyjádření společného
- Vytvoření modifikovatelných specifikací
- Opětné použití analytických výsledků
- Konzistentní nosná reprezentace pro analýzu a návrh

Principy zvládnutí složitosti

- abstrakce procedurální a datová
- zapouzdření
- dědičnost
- sdružování
- komunikace pomocí zpráv
- postup metody organizace
- měřítko
- kategorie chování

Abstrakce

- principem abstrakce je ignorovat ty aspekty subjektu, které nejsou významné pro současný účel, abychom se mohli víc soustředit na ty, které významné jsou.
- vymezuje podstatné znaky objektu, které jej odlišují od ostatních druhů objektů, a tak poskytuje ostře definované koncept. hranice relativně podle perspektivy pozorovatele.



- **Procedurální abstrakce:** každá operace, která docílí jasně definovaného výsledku, může být považována a používána jako jednoduchá entita, i když ve skutečnosti je realizována nějakou sekvencí operací nižší úrovně.
- **Datová abstrakce:** princip definice datového typu pomocí operací, které jsou aplikovány na objekty příslušného typu s tím omezením, že hodnoty těchto objektů mohou být modifikovány a pozorovány pouze pomocí operací.

Zapouzdření

- ukrytí informace
- princip použitý při vývoji celé programové struktury
- spočívá v tom, že každá komponenta programu by měla zapouzdřit a ukrýt jediné návrhové rozhodnutí.
- rozhraní každého modulu je definováno tak, aby odhalilo co nejméně o vnitřním dění v modulu.

Objekt

- má stav, chování a identitu
  - stav objektu zahrnuje všechny (obvykle statické) vlastnosti objektu plus současné (obvykle dynamické) hodnoty těchto vlastností
  - chování vyjadřuje, jak objekt koná a reaguje, v pojmech změn stavu a předávání zpráv
- struktura a chování podobných objekt jsou definovány v jejich společné třídě
- pojmy instance a objekt jsou vzájemně zaměnitelné

Třída

- je množina objektů, které sdílejí společnou strukturu a shodné chování
- je abstraktní datový typ vybavený možnou částečnou implementací
- objekt je instancí třídy
- atributy - reprezentují strukturu třídy

Hierarchie

- znamená hodnostní zařazení nebo uspořádání abstrakcí
- 2 nejvýznamější hierarchie:
  - „is-a” hierarchie („je něčím”) = dědičnost → např. medvěd je savcem
  - „part-of” hierarchie („součást něčeho”) = sestava vnitřní struktury objektu → např. převodovka je součástí auta

Dědičnost

- je vztah mezi nadtřídou a jejími podtřídami
- mechanismus, který vyjadřuje podobnost mezi třídami
- zjednodušuje definici třídy pomocí již dříve definované třídy
- vyjadřuje generalizaci (zobecnění) a specializaci tím, že v hierarchii tříd explicitně určuje společné atributy a služby

Propojení (vazba)

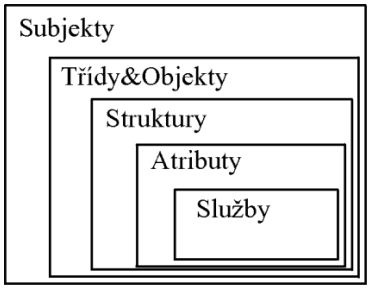
- fyzické nebo konceptuální spojení mezi objekty
- označuje specifickou asociaci, pomocí níž 1 objekt (klient) používá služby jiného objektu (poskytovatel, dodavatel) nebo pomocí níž může jeden objekt navigovat druhý objekt
- Role účastníků propojení
  - Aktor (aktivní objekt) - může operovat s jinými objekty, ale sám není nikdy takto operován
  - Server - nikdy neoperuje s jinými objekty, a sám je operován jinými
  - Agent - operuje s jinými objekty a sám je operován jinými objekty (agenty i aktory)

Polymorfizmus umožňuje:

- jednomu objektu volat jednu metodu s různými parametry (parametrický polymorfismus)
- objektům odvozených z různých tříd volat tutéž metodu se stejným významem v kontextu jejich třídy, často pomocí rozhraní
- přetěžování operátorů znamená provedení operace v závislosti na typu operandů (operátorový polymorfismus)

Coad&Yourdon: 5-ti vrstvý model

1. identifikace tříd a objektů (dědičnost)
2. identifikace struktur („part-of” vztahů)
3. definice subjektů
4. definice atributů



5. definice služeb

UML

- Unified Modeling Language
- standardní jazyk pro zobrazení, specifikaci, konstrukci a dokumentaci artefaktů systémů s převážně SW charakterem
- může být použit při všech procesech životního cyklu vývoje a pro různé technologie a implementace

UML umožňuje (mimo jiné):

- zobrazit hranice systému & jeho hlavních funkcí pomocí případů užití (use cases) a účastníků (actors)
- ilustrovat realizaci případů užití pomocí diagramů interakcí
- reprezentovat statickou strukturu systému pomocí diagramů tříd
- modelovat chování objektů pomocí stavově-přechodových digramů
- odhalit fyzickou implementační architekturu pomocí diagramů zapojení komponent
- rozšířit funkcionalitu pomocí stereotypů

Modelovací techniky UML

- specifické vyjadřovací prostředky (jazyky) pro popis konceptů na vysoké úrovni abstrakce
- grafické vyjádření – pro pochopení, dokumentaci a vysvětlení problému
- zápis myšlenek strukturovaným způsobem napomáhá ke zvládnutí složitosti a multidimenzionality SW

Diagramy v UML

1) modely ukazující statickou strukturu systému

- Diagramy struktury definují statickou architekturu modelu. Jsou používané na modelovaní prvků, z nichž se systém skládá (třídy, objekty, rozhraní a fyzické komponenty). Používají se pro modelování relací a závislostí mezi jednotlivými elementy.
- diagramy tříd (grafický pohled na statickou strukturu) a objektové diagramy (instance diagramu tříd, ukazuje snímek systému v určitém bod)
- implementační diagramy (diagram komponent, rozmístění)

2) modely ukazující dynamické chování systému

- Zachycují interakce a okamžité stavy včetně chování v čase. Popisují reakce systému v reálních podmínkách a sledují vlivy operací a událostí včetně výsledků.

- model případů užití - externí pohled na systém (aktéři a případy užití)
- diagram aktivit - externě/interní pohled na systém
- interakční diagram: diagram sekvencí a diagram spolupráce - interní pohled na systém

3) modely ukazující dynamické chování jedné třídy

- stavové diagramy, diagramy aktivit

1) Modelování požadavků na systém

- **use case diagram (diagram případů užití)**: aktéři (role–uživatelé, externí systémy i čas) a případy užití (funkcionalita)
- specifikace případů užití pomocí **minispecifikace** – název, aktéři, omezení, toky událostí (scénáře). Forma: číslované kroky, pseudokód, **diagram aktivit** nebo pomocí **diagramů interakcí** (typicky zachycují chování jednoho případu užití. Patří sem sekvenční diagramy/diagramy posloupností a diagramy spolupráce) s pseudokódem.

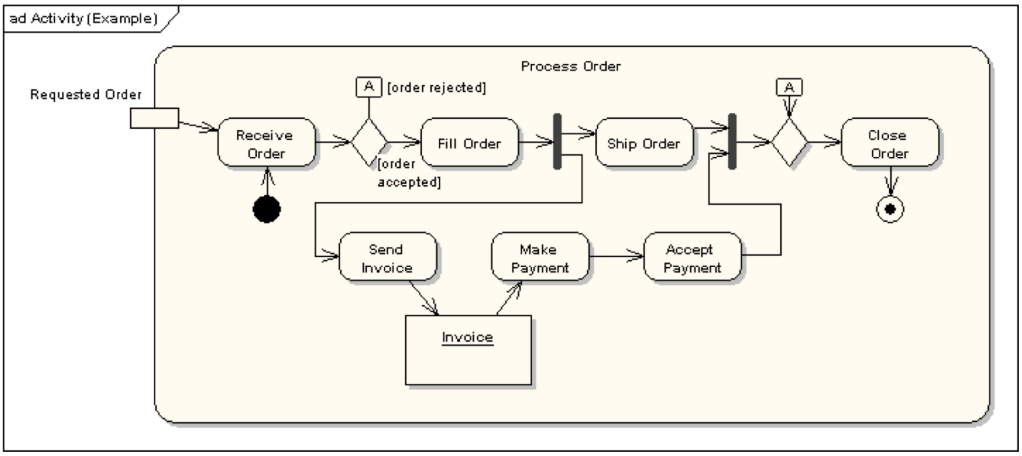


Diagram aktivit

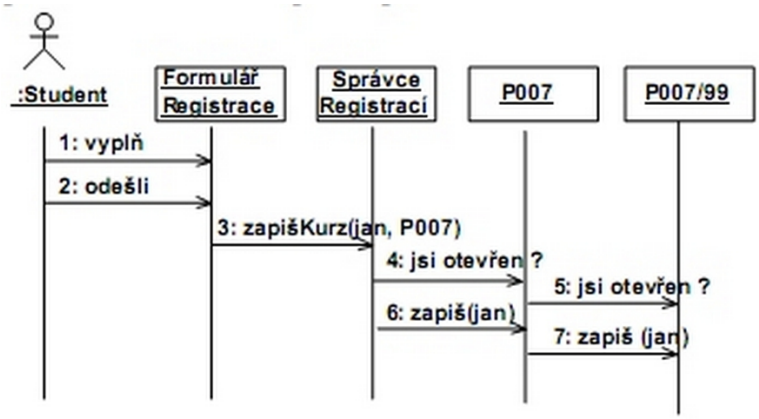


Diagram posloupnosti (Sequence diagram) ukazuje interakci mezi objekty uspořádanou do časové posloupnosti.

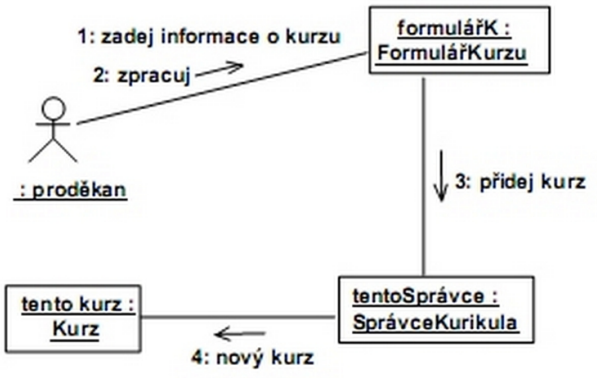


Diagram spolupráce (Collaboration diagram) ukazuje interakce objektů a jejich propojení mezi sebou.

2) Analýza, modelování analytických tříd a objektů

- základní modely, bez implementačních detailů
- většinou platformě nezávislé
- **diagramy tříd** (grafický pohled na statickou strukturu): atributy, vztahy mezi třídami (asociace, závislosti, generalizace/specializace)
- **objektové diagramy** (instance diagramu tříd, ukazuje snímek systému v určitém bodu/času)
- **diagram balíků**: balíky na analytické úrovni

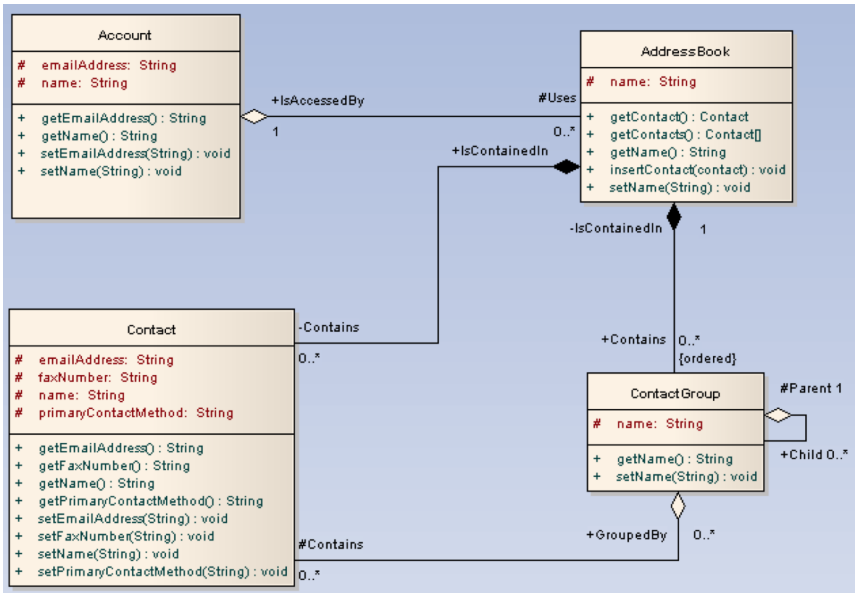
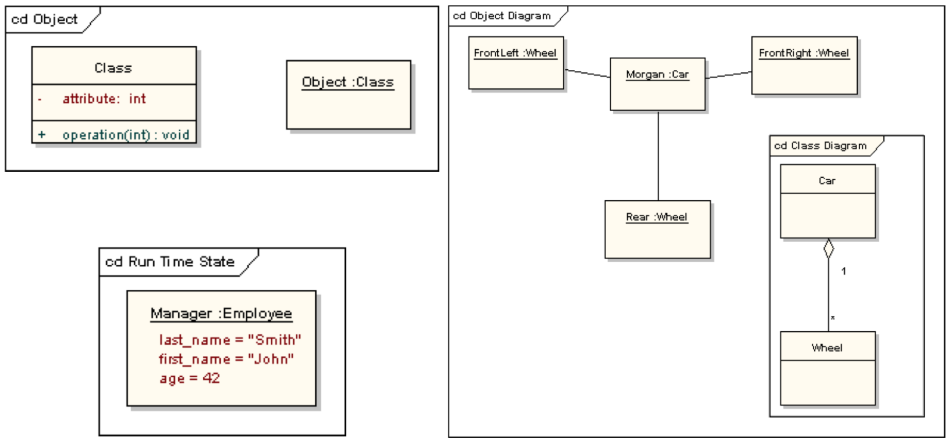


Diagram tříd



Objektový diagram

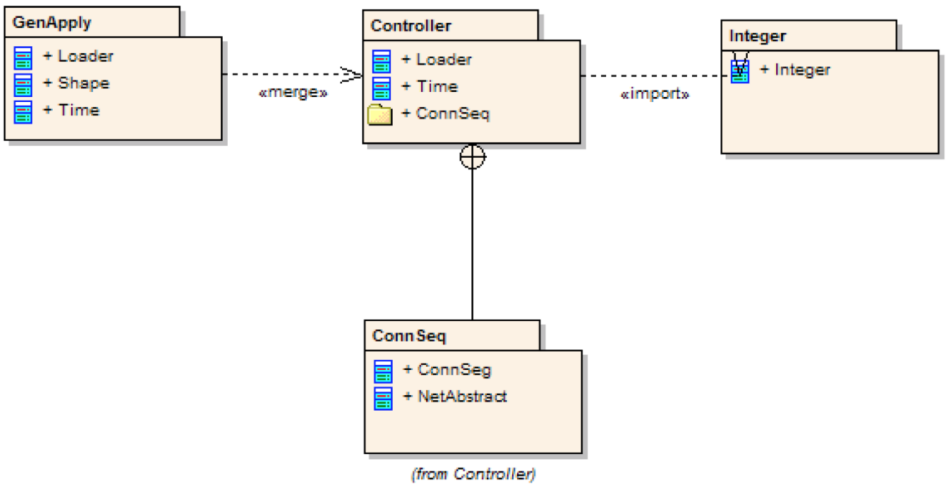
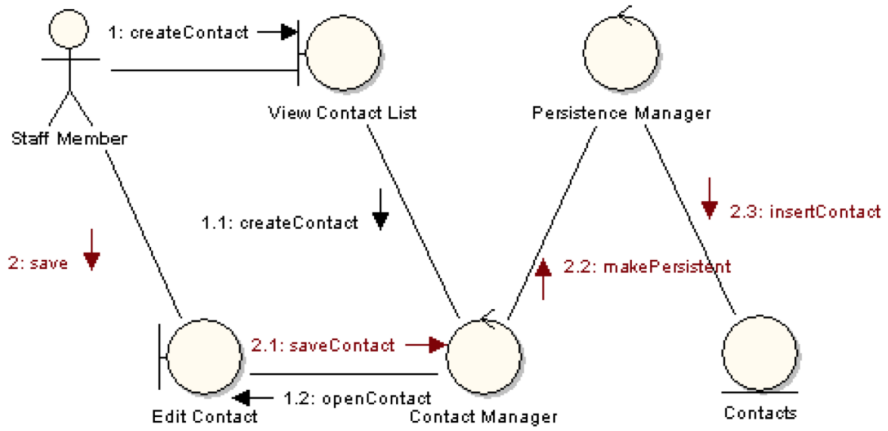


Diagram balíku (package diagram)

3) Realizace případů užití (analýza/návrh)

Diagramy interakcí:

- **sekvenční diagramy** (sequence d.) – modelují výměnu zpráv s důrazem na časovou osu (teď už tam jsou přímo objekty, předtím tam byly „reálný věci“)
- **komunikační diagramy** (communication d.) – modelují interakce organizované podle interagujících objektů
- **diagramy přehledů interakcí** (interaction overview d.) – ukazují realizaci složitého chování pomocí několika jednodušších interakcí (kombinuje sekvenční diagramy a digramy aktivit). Jde o formu diagramu aktivit. Každý „obdélník“ reprezentuje jeden diagram aktivit.
- **časovací diagramy** (timing d.) – zaměřují se na “real-timové” aspekty interakcí, časová omezení a závislosti, … Zobrazuje změny stavů nebo hodnot elementů v čase.



Komunikační diagram

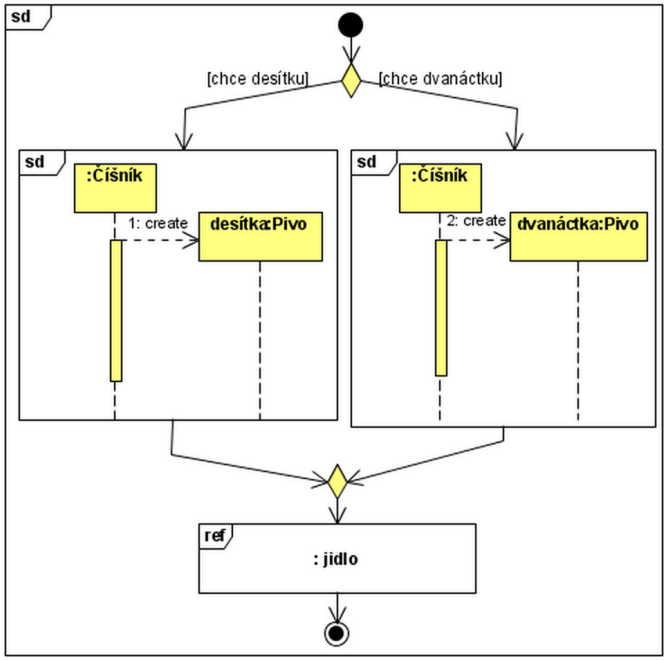


Diagram přehledu interakcí

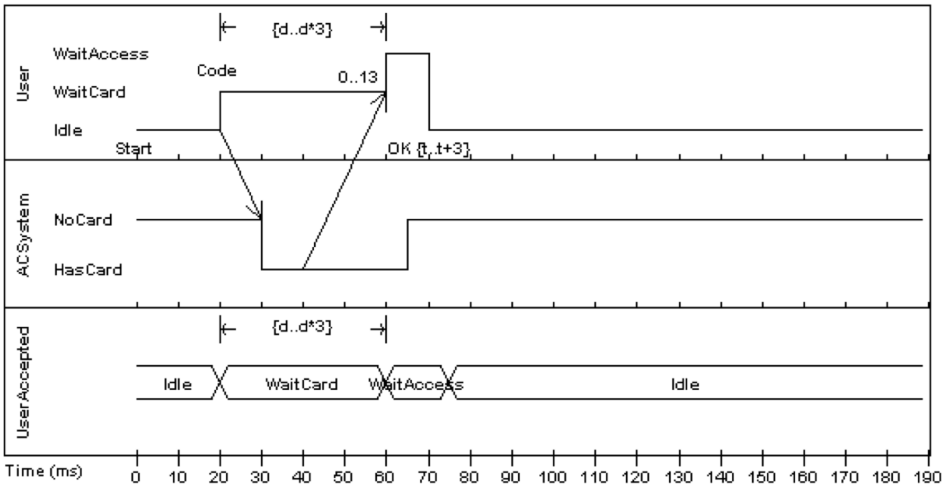


Diagram časování

Návrh

- detailní modely, zvolený cílový jazyk, technologie, ...
- detailní **diagram tříd** na návrhové úrovni: návrhové třídy (otypované atributy a metody), vazby (upřesnění asociace: směr, agregace, kompozice, násobnost)
- **diagram balíků**: balíky na návrhové úrovni
- **diagramy komponent**: Ilustruje základní části softwaru, které tvoří systém. Poskytuje vyšší úroveň abstrakce než diagram tříd, obvykle je komponenta implementována pomocí jedné nebo více tříd.
- **diagramy rozmístění (nasazení):** Modeluje architekturu systému v reálnem provozu. Ukazuje konfiguraci hardwarových komponent (uzlů) a také jak jsou softwarové prvky a artefakty na tyto hardwarové komponenty namapované.
  - Uzel je buď hardwarový nebo softwarový prvek reprezentovaný jako kvádr. Instance uzlu – může obsahovat i instance uzlů.
  - Instance uzlu má na rozdíl od uzlu podtrhnuté jméno a dvojtečku před základním typem uzlu.

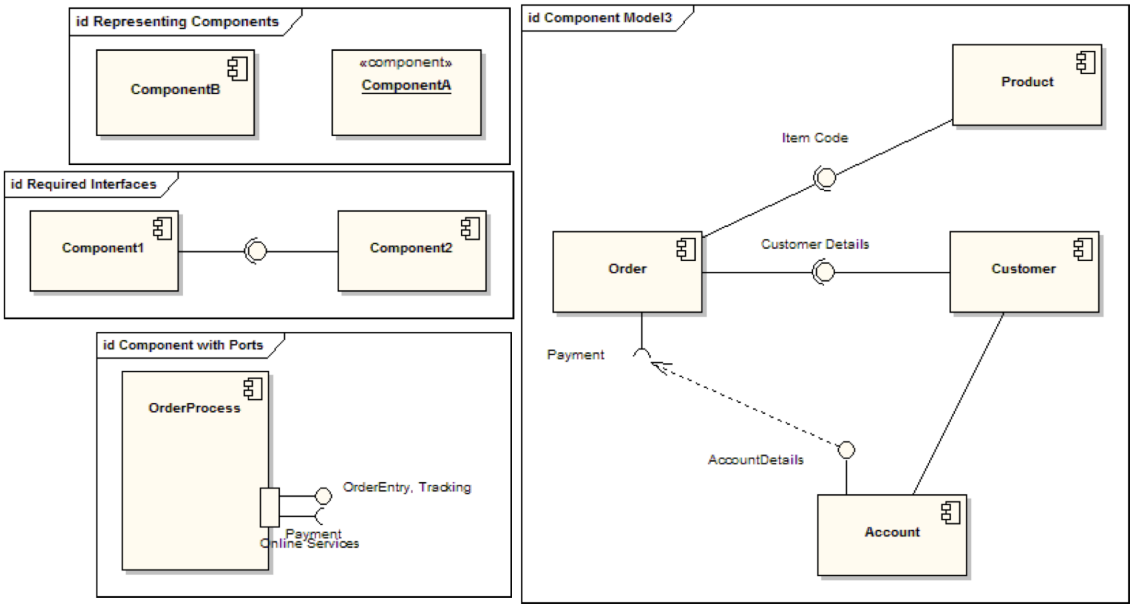


Diagram komponent

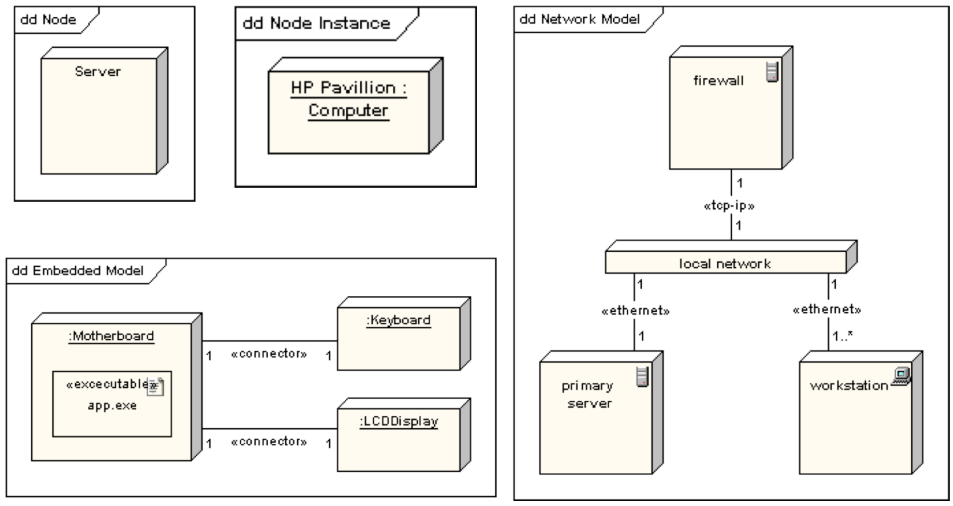


Diagram nasazení (deployment diagram)

Vztahy

- cesta pro komunikaci mezi objekty.
- **asociace -** obousměrné propojení mezi třídami
- **agregace -** vztah mezi celkem a jeho částmi (silnější forma vztahu)
- **kompozice** – agregace se silnějším vlastnictvím a společným životem částí v celku. Části neexistují vně celku, každá část patří do jediného celku.
- **závislost** - slabší forma vztahu, mezi klientem a poskytovatelem, klient nemá žádnou znalost o poskytovateli

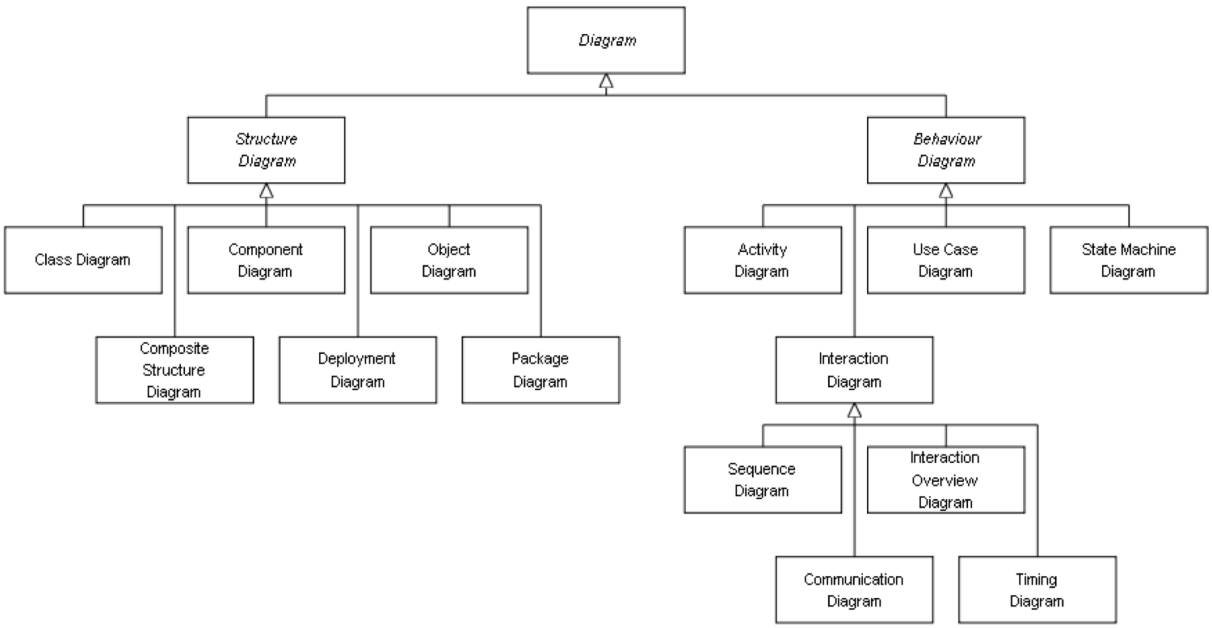
Násobnost → definuje, kolik objektů se účastní vztahů (počet instancí jedné třídy vztažený k jedné instanci druhé třídy)

Navigace → určuje směr propojení a směr návěstí

Dědičnost → vztah mezi nadtřídou a podtřídami

Událost → z pohledu stavového diagramu je to výskyt jevu, který může spustit přechod mezi stavy

Modely zahrnuté v UML:



3. Nástroje a modely datové, funkční a časové dimenze systému

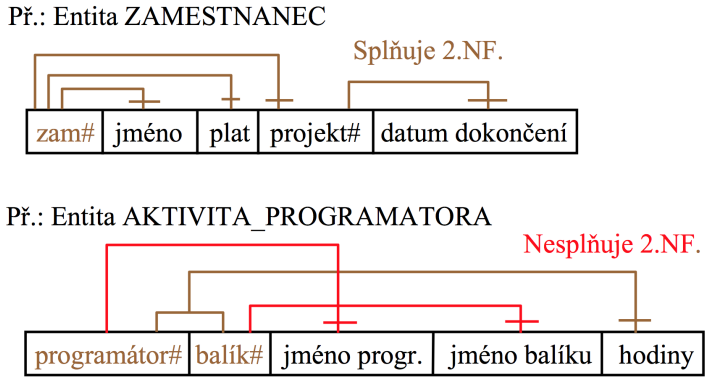
Strukturovaná analýza

Datové dimenze

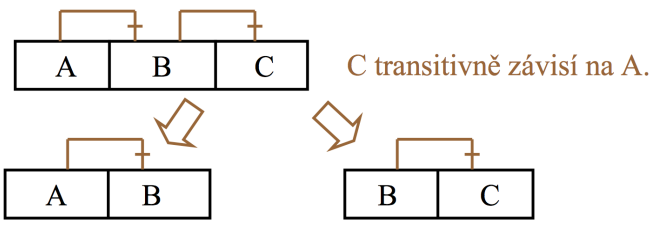
- Datový model:
  - definuje neměnné atributy a strukturu dat. Slouží jako stabilní základ procesního modelu. Vyjadřuje vztahy, které nejsou zachyceny v procesních modelech.
- Komponenty datového modelu:
  - entita (objekt, o němž uchováváme informace), entitní množina (množina entit stejného typu/druhu), vztah (vztah mezi entitami, který evidujeme a o němž uchováváme informace),vztahová množina (množina vztahů stejného typu/druhu), atribut (vlastnost entity nebo vztahu, jejíž hodnotu chceme uchovat a používat v systému), doména atributu (obor hodnot atributu)
- ERD (diagram entit a vztahů)

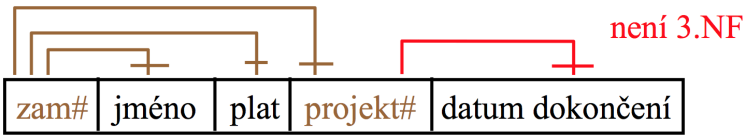
  [vsuvka]

  - 1. NF: Datový záznam je v 1. NF, pokud všechny jeho komponenty jsou atomické. Datový záznam je v 1. NF pokud se v něm nevyskytují opakující se skupiny položek
  - 2. NF: Požaduje plnou funkční závislost všech neklíčových atributů na celém klíči. Problémy (Dokud nám dodavatel nedodá součást, nemůžeme zapsat jeho adresu a další údaje. Pokud přestane dodavatel dočasně zásobovat, pak zrušení záznamu o součásti zruší i jeho údaje. Jakákoliv změna v údajích o dodavatelích je komplikovaná - vyhledání a oprava více záznamů.) se dají vyřešit „rozbitím“ do více tabulek



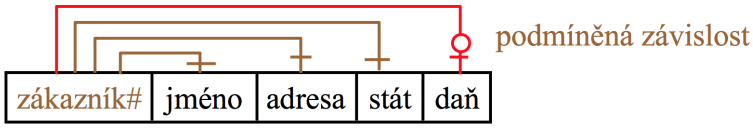
  - 3. NF (tranzitivní závislost): Záznam R je ve 3.NF, pokud je ve 2.NF a každá neklíčová položka R je netranzitivně závislá na každém kandidátním klíči z R.



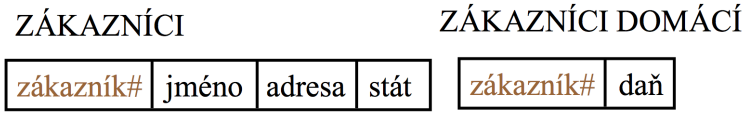


- 4. NF (podmíněná závislost)

  4.NF odstraňuje podmíněné funkční závislosti.



  Daň strháváme těm, kteří sídlí ve stejném státě jako naše firma.



- datový slovník (Data Dictionary)
  - jedná se o seznam definicí datových prvků systému.
  - popisuje obsah dat v datových tocích a pamětech na DFD, entity a atributy na ERD i další klíčová slova.

Příklad datového slovníku (DD) popisující datový tok objednávka zboží:

objednávka zboží = @ id objednávky + kód zákazníka
+ {název zboží + množství zboží}
id objednávky = {0 – 9}
kód zákazníka = {0 – 9}
název zboží = {{a – ž} | {A – Ž}}
množství zboží = {0 – 9}

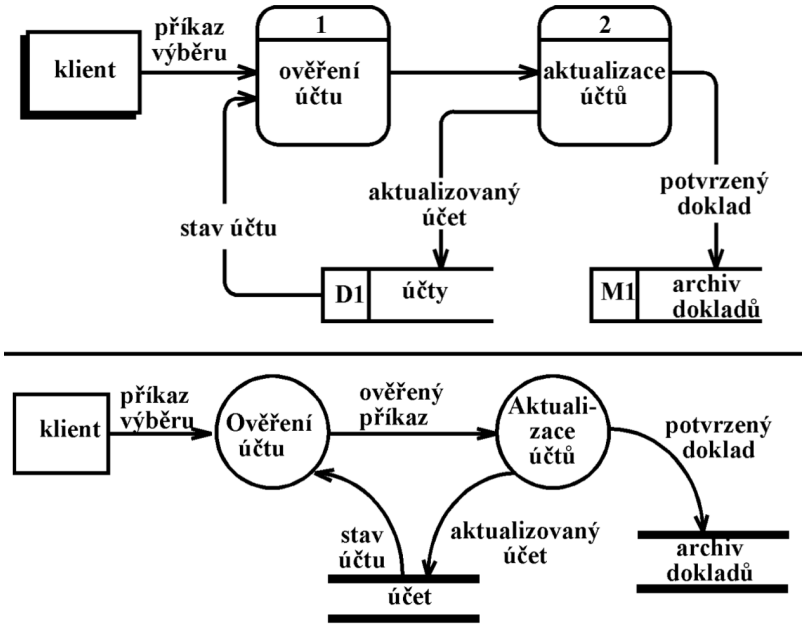
Notace DD:

- = je 'skládá se z'
- + je 'a'
- @ je klíč
- {...} značí opakující se prvek
- (...) nepovinný prvek

Funkční dimenze

- DFD
  - je modelovací nástroj, který umožňuje zobrazit systém jako síť procesů, které plní určené funkce a předávají si mezi sebou data. Poskytuje funkčně orientovaný pohled na systém.
  - datové toky (cesty, po kterých se pohybují datové shluky – informační pakety – z jedné části systému do druhé)
  - procesy (části systému, které transformují určité vstupy na vstupy)
  - paměti (kolekce dat v klidu)
  - terminátory (externí entity, se kterými systém komunikuje)
- DFD bývají víceúrovňové
- minispecifikace procesů
  - slovní popis algoritmického chování (každý proces má minispecifikaci nebo je dále dekomponovaný, jiná varianta není). Je nástrojem analýzy a návrhu a je

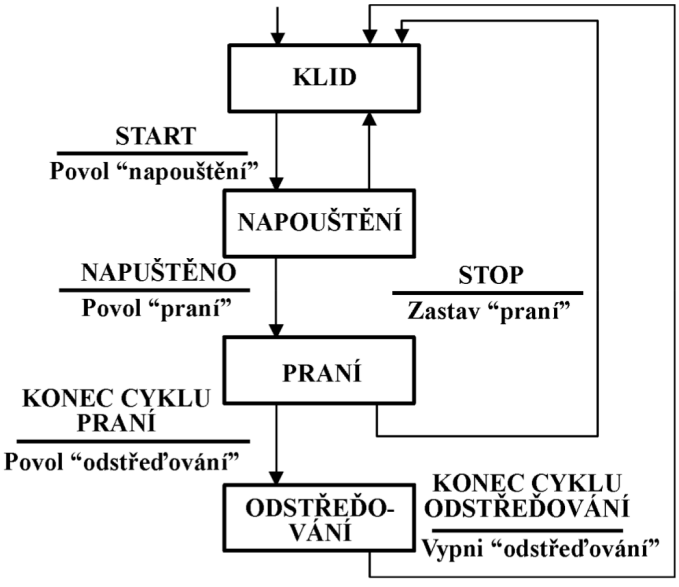
konzultovaná se zákazkíkem.



Data Flow Diagram

Časové dimenze

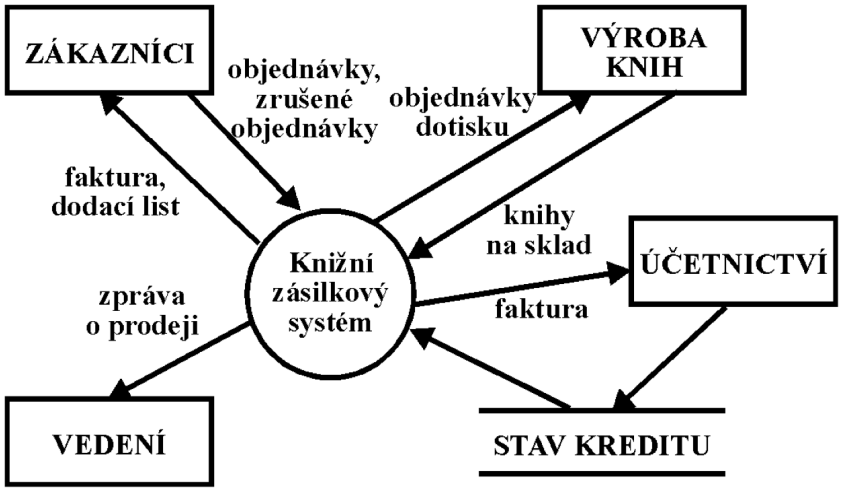
- STD
- graf stavů a přechodů



Stavový diagram (STD)

Souhrn nástrojů strukturované analýzy:

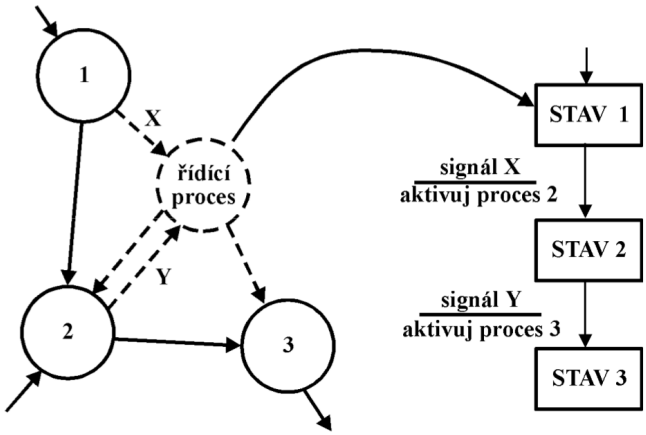
- Kontextový diagram
- Diagram datových toků - DFD
- Minispecifikace procesů
- Datový slovník - DD
- Entitně-relační diagram - ERD
- Stavový diagramy - STD
- Řídící DFD - CDFD



Kontextový diagram je zvláštním případem DFD. Obsahuje jediný proces, který reprezentuje celý systém.

CDFD - Control Data Flow Diagram

- DFD rozšířený o řídící procesy, toky a paměti.



Každý řídící proces je popsán pomocí STD. Vstupní signály se uplatňují v podmínkách přechodů, akcím přechodů odpovídají procesy na CDFD.

Objektová analýza

Datové dimenze

- statické diagramy

Funkční + časové dimenze

- dynamické diagramy
- stavový diagram

Jaké modely se používají:

	Strukturovaná analýza	**Objektová analýza**
Okolí systému:	Kontextový diagram	Use case diagram
Funkcionalita:	Diagram datových toků Diagram datových toků s řízením	Sekvenční diagram Diagram spolupráce Diagram činností
Data:	Entitně relační diagram	Diagram tříd Diagram objektů
Řízení:	Stavový diagram	Stavový diagram (rozšířený)

4. Softwarové metriky

Při řízení prací při vývoji či customizaci a také při provozu IS je nutné sledovat kvantitativní (číselné) charakteristiky, jako je počet řádků programů, doba řešení, pracnost řešení atd., umožňující hodnotit průběh prací a odhadovat kvalitu softwaru a přijímat odpovídající opatření.

Measure

- Quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process
- Number of errors

Metric

- Kvantitativní charakteristiky softwaru
- Quantitative measure of degree to which a system, component or process possesses a given attribute. „A handle or guess about a given attribute“.
- Number of errors found per person hours expended

Charakteristika:

- bez měření nelze kvalitně řídit ani hodnotit kvalitu SW, atributy kvality mohou ale být různé
- je součástí business intelligence SW firmy a přepoklad uplatnění moderních způsobů řízení (CMM)
- sledování kvantitativní charakteristiky (počet řádků, doba řešení, spotřeba práce, ...)
- základ zajištění kvality SW
- hodnoty metrik jsou cosi jako paměť firmy
- metrika = „kvalifikovaný” atribut SW

	Del	*Noper*	*Nrnd*	*Soper*	*Srnd*
`begin`	1	1		1	
`var x,y:real;`	7	5	2	5	2
`x:= y*2.0+sin(x)+2.0`	12	7	5	5	1
`end.`	1	1		1	
Celkem	21	14	7	12	3

Tab. 15.1: Příklad výpočtu hodnot metrik jednoduchého programu v jazyce Pascal.

Použití SW metrik

- **Výzkum:** podklad pro hledání metod realizace softwarových produktů, které by přinesly podstatné zvýšení jeho kvality a snížení nákladů a doby vývoje a hlavně rozsahu prací při údržbě softwaru (výzkum metod a zákonitostí vývoje softwaru).
- **Normy:** základ pro stanovení technicko-ekonomických podkladů pro řízení prací při tvorbě softwaru (normy pracnosti, odhady takových metrik, jako je pracnost či doba

řešení) a uzavírání smluv (cena, termíny), předpoklad CMM.

- **Kontrola kvality:** prostředek sledování spolehlivosti softwaru při provozu a podklad pro řídící zásahy během údržby,procesy zajišťující kvalitu.
- **Operativa:** prostředek sledování průběhu prací při vývoji (dodržování termínů, procento testovaných komponent, trendy počtů chyb, počty nově zanesených chyb, komponenty s největším počtem chyb, atd.).

Druhy metrik

- **Implicitní (in proces)** – zjistitelné pouze během vývoje
  - Prac – pracnost
  - Doba – doba řešení
  - team(t) a Team – velikost týmu v čase a průměrná velikost
  - MTBF(t) – střední doba mezi poruchami
  - další: spotřeba práce, produktivita, počet selhání systému v čase, defekty, spokojenost zákazníka, ...
- **Explicitní (after proces)** – zjistí se kdykoliv i po skončení vývoje (z artefaktů produktů systému)
  - Del – délka programu v řádcích. Základ pro COCOMO
  - Fun(P) – počet podprogramů, počet metod
  - Srnd a Nrnd – rozsah a výskyt operandů. Operand je bud'konstanta (např. celé číslo 10, nebo řetězec znaků "xyz", v terminologii programování literál), nebo proměnná, např. *x*. *Srnd* je pak počet logicky odlišných operandů vyskytujících se v programu
  - Soper a Noper – rozsah a výskyt operací,
  - další: počet funkcí, tříd, počet datových tabulek, ...
- **Interní** – potřebuje řešitelský tým pro řízení a kontrolu prací: cost, effort, LOC, speed, memory, ...
- **Externí** – charakterizují uživatelské vlastnosti produktu: functionality, quality, complexity, efficiency, reliability, maintainability, ...
- **produktové:** zdrojový kód, dokumentace, cyklomatická složitost (počet cest programem), počty funkčních bodů
- **procesní:** činnosti spojené s vývojem, doba strávená na jednotlivých úlohách, původní odhad a skutečná reálná doba
- **metriky zdrojů:** HW, lidé, čas, nemocnost, výkonnost

Implicitní metriky jsou pro řízení prací na vývoji či customizaci softwaru nejdůležitější. Základním problémem řízení projektu je odhad implicitních metrik, jako je cena, pracnost a doba řešení. Nejdůležitější implicitní metriky jsou:

- Pracnost realizace (špotřeba člověko-měsíců)
- Doba (měsíce)
- Produktivita (počet jednotek délky za člověko-měsíc)
- Průměrná velikost týmu,
- Počet selhání v čase *t*
- Počet defektů v čase *t*
- Míra spokojenosti zákazníků v čase *t*

Potíže s metrikami

- rozptyl hodnot
- produktivita práce programátor, jejich kvalita
- druh SW, obtížně splnitelné termíny realizace
- moderní projekční techniky
- kvalita zúčastněných
- omezení SW a HW

Datové typy metrik

- příslušnost ke třídě – id, číslo tramvaje; vztah: =
- fuzzy metrika - slabý, dobrý, velmi dobrý, vynikající; vztah: =, <
- fuzzy číselná – známky ve škole; vztah: =, < , (aritm. operace)
- interval – teplota, čas
- číselná metrika – plná data (délka, ...)

5. CMM

CMM = Capability Maturity Model

- definuje kvalitu procesu, ne jen výrobků
- neříká jak, ale pouze co je nutné mít
- cílem je zvýšení spokojenosti uživatelů SW systémů, zlepšení kvality SW a omezení rizik spojených s vývojem SW
- nástroj na zlepšení efektivity práce firmy, obsahuje postupy na snižování nákladů, zkrácení termínů, eliminaci rizik spojených s migrací pracovníků
- hodnotí vyspělost organizací podle stupně a kvality využívání SW procesů (SWP)
- definuje 5 úrovní vyspělosti SWP

1. Počáteční úroveň (*i*nitial level)

Chaotický proces, nepředvídatelná cena, plán a kvalita.

- SWP v neformální formě a definuje se případ od případu od počátku
- nejsou pevná pravidla plánování a řízení projektů
- výsledky záleží spíš na kvalitě jednotlivce než organizaci práce, zkušenosti se nevyužívají, po odchodu pracovníka jsou ztraceny

2. Úroveň zajišťující opakovatelnost (**r**epeated level)

Intuitivní cena a kvalita jsou vysoce proměnlivé. Neformální metody a procedury. Zavedena pravidla pro řízení projektu, plánování a řízení založeno na zkušenostech. Jednotné zásady pro celou firmu, SWP nejsou standardizovány, plány realizace se sledují, náprava při odchylkách.

- řízené požadavky
- plánování softwarového projektu
- řízené subkontrakty na software
- zajištění kvality software
- řízení softwarových konfigurací

3. Úroveň definovaných procesů (**d**efined level)

Orientován na kvalitu. Spolehlivé ceny a plány, zlepšující se, ale dosud nepředvídatelný přínos (výkon) systému kvality. SWP standardizováno (jak procesy SW – inženýrské, tak manažerské), součástí norem jsou nástroje kontroly a zvýšení efektivity práce, zkušenosti a osvědčené metody a postupy. Součástí standardů jsou procedury přizpůsobení SWP na konkrétní projekt, zajištěna kontrola dodržování požadavků, nákladů a termínů. SWP založeno na odborném zázemí a znalostech pracovníků firmy, pravidelná školení.

- zlepšování organizačního procesu
- definice organizačního procesu
- standardizace prací všech etap vývoje

- školicí program
- řízení integrovaného software
- aplikace inženýrských metod u softwarového produktu
- podpora týmové práce, spolupráce mezi týmy
- detailní prověrky a oponentury, audity

4. Úroveň řízení procesů (**c**ontroled level)

Kvantitativní; promyšlená statisticky řízená kvalita produktu. Definovány metriky kvality pro SWP i vývoj SW, systém sběru, sledování a vyhodnocování metrik jednotným způsobem v rámci celé organizace. Firma je schopná vyhodnocovat trendy a odhad hodnoty důležitých metrik, schopná odhadnout přesnost odhadů stanovením konfidenčních intervalů (intervaly spolehlivosti), tj. mezí, do nichž s velkou pravděpodobností padne odhadovaná hodnota.

- měření a kvantitativní řízení procesu výroby
- řízení kvality

5. Úroveň optimalizace procesů (**o**ptimized level)

Kvantitativní základ pro kontinuální investice směřující k automatizaci a zlepšení výrobního procesu. Zavedeny procedury neustálého vylepšování SWP, vytvořen tým hodnotící kvalitu procesů a navrhující vylepšení včetně zavádění nejnovějších metod, postupů a nástrojů. Tým analyzuje příčiny úspěchů i neúspěchů, pak modifikuje SWP.

- prevence chyb
- inovace technologie
- optimalizace SW procesů
- řízené změny výrobních procesů

[Mnemotechnická pomůcka pro zapamatování anglických názvů CMM úrovní: **IR**i**D**ium **CO**balt]

6. Odhady COCOMO a funkční body

Odhady

- dva principy odhadu – odhady pracnosti E (effort, člověko-měsíce) a doby řešení T (time, měsíce)
- obě varianty odhadu (COCOMO, FP) jsou použitelné v různých etapách životního cyklu
- COCOMO → vychází z odhadu délky programů KSLOC = tisíce zdrojových řádků (COCOCMO II už může mít na vstupu UFP)
- Funkční body → odhad na základě struktury aplikace (nezkoumá kód ale funkci aplikace)

COCOMO

- COCOMO – **CO**nstructive **CO**st **MO**del
- existuje víc druhů COCOMO – COCOMO 81, COCOMO II

COCOMO 81

- Předpoklady, myšlenky:
  - Cena vývoje aplikace přímo závisí na velikosti SW.
  - Přesnost odhadu velikosti SW závisí na etapě vývoje.
    - V pozdějších etapách je odhad přesnější.
    - Přesnost odhadu se může lišit až čtyřikrát (4:1) oběma směry (kužel nejistoty), např. 25 000 – 100 000 – 400 000
- 3 úrovně detailu (čím máme přesnější data, tzn. čím jsme dál ve vývoji) tím používáme přesnější model (tzn. zohledňujeme více faktorů)
  - **Základní model** – hrubý odhad E(KSLOC) a T(KSLOC) založen pouze na odhadu KSLOC.
  - **Střední model** – vliv jiných faktorů na E(KSLOC) a T(KSLOC). Jde o tzv. korekční faktory (zkušenost týmu, kvalita SW, ...)
  - **Pokročilý model** – bere v úvahu vlivy vývojové etapy (návrh, implementace, ...), ve které se projekt nachází.
- Potřebné údaje získáváme často z předchozích projektů (projekty nejsou většinou totožné a tak je potřeba údaje upravit)
- Vývojové módy projektů
  - **Organický mód** – velikost projektu = do 50 000 LOC, malé problémy pro malé týmy, typické jsou mírné normy a malá omezení na specifikaci rozhraní a možnost ovlivnit požadavky. Algoritmy a postupy jsou dobře známy, podmínky realizace relativně stabilní, nejsou ostré podmínky ani termíny. Příklad: zpracování dat v dávkovém provozu, úpravy známého operačního systému nebo kompilátoru, jednoduchá skladová aplikace

    → všemu perfektně rozumín, bude to „brnkačka“, už jsem to dělal tisíckrát

- **Bezprostřední (přechodný) mód** – typ mezi organickým a vázaným, velikost projektu < 300 KSLOC, pokud není kód generován automaticky. V týmu jsou zkušení i méně zkušení pracovníci, tým má nemoc velké zkušenosti z předchozích obdobných realizací, úloha poměrně složitá. Příklad: menší dedikované oper. systémy, běžné transakční systémy, systém řízení výroby, jednoduchý systém řízení vojsk a zbraní, nový operační systém a překladač, středně složitá skladová aplikace. Sem spadá většina projektů.

  → ccelkem hápu požadavky, mám zkušenosti, ale něco nového si taky vyzkouším

- **Vázaný mód** – SW pracuje za velmi ostrych omezení na výkonnost a dobu odezvy, velikost jsou miliony řádků, vyžaduje práci s komplikovanými SW a HW systémy za ostrých předpokladů na předepsané fce, spolehlivost, termíny, přenositelnost, modifikovatelnost. Požadavky je obtížné měnit, řeší se nové problémy, se kterými se pracovníci dosud nesetkali. Obvykle: specifikaci požadavků + návrh dělá malá skupina, vývoj částí – velký tým. Programování a testy – souběžné. Příklad: nové rozsáhlé oper. systémy, kosmické lodě, letadla, atomové elektrárny, RT aplikace. Model výzkumník apod.

  → tuším co dělat, ale velká omezení, nové algoritmy, spec. HW

- Atributy produktu
  - RELY – požadovaná spolehlivost
  - DATA – velikost databáze
  - CPLX – složitost produkt
- HW atributy
  - TIME – omezení času výpočtu
  - STOR – využití paměti/disku
  - VIRT - spolehlivost virtuálních stroj
  - TURN – míra rychlosti oběhu úlohy počítačem
- Atributy vývojového týmu
  - ACAP – analytické schopnosti
  - PCAP – programovací schopnosti
  - AEXP – zkušenosti s podobnými aplikacemi
  - VEXP – zkušenosti se spec. virt. strojem
  - LEXP – zkušenosti se spec. prog. Jazykem
- Atributy projektu
  - MODP – použití moderních prg. technik
  - TOOL – použití SW nástrojů
- Vzorce

$$E = a \times (KSLOC)^{b}$$

$$T = c \times E^{d}$$

a, b, c, d : parametry podle úrovně modelu a vývojového módu

- V základním modelu mají všechny parametry konstantní hodnoty.
- Ve středním a pokročilém modelu ve všech vývojových módech *a* závisí na F_c *(a x* F_c*)* ostatní parametry jsou konstantní.
  - Korekční faktor F_c je součinem hodnot 15 atributů specifických pro vývojový proces
- funguje až od 2 000 LOC, pro menší hodnoty dává špatné výsledky

- Postup při stanovení odhadu pracnosti
  1. určí se (odhadne) úroveň modelu a mód projektu (*a* a *b* bereme z tabulky) → nominální úsilí E_n
  2. určí se korekční faktor → hodnocení vlivu faktorů reprezentovaných jednotlivými 15ti atributy (velmi nízký, nízký, normální, velký, velmi velký, extrémně velký). Na základní úrovní se neřeší.

     $F_c = suma_{i=1...15}(F_i)$. Zpravidla se F_c, může ale zůstat 1.
  3. určí se aktuální (zpřesněné) úsilí *E* (člověk-mšsíc) → na základní úrovni $E = E_n$, jinak $E = F_c * E_n$
  4. určí se doba vývoje $T \rightarrow T = c * E^d$ (*c* a *d* bereme z tabulky)

COCOMO II (1995)

- Potřeba změnit COCOMO 81
  - nové softwarové procesy
  - nové jevy měření velikostí
  - nové jevy znovupoužití software
  - potřeba rozhodování na základě neúplné informace
- stejný princip výpočtu, stejné vzorce, jen jinak korekční faktor
- 3 různé modely
  - ACM (Aplication Composition Model) pro projekty s použitím moderních nástrojů a GUI
  - EDM (Early Design Model) pro hrubé odhady v úvodních etapách, kdy se architektura vyvíjí
  - PAM (Post Architecture Model) pro odhady poté, co byla specifikována architektura
- Výpočet

$$PM_{estimated} = A \times (Size)^{(SF)} \times \left(\prod_i EM_i \right)$$

$$SF = 1.01 + 0.01 \times \sum (hodnocení\ driverů\ exponentu)$$

A v rozsahu 1.01 – 1.26

  - Size je určená několika přístupy:
    - KSLOC (tis.řádek zdroj. kódu)

        - UFP (neupravené fukční body)
        - EKSLOC (ekvivalentní velikost zdroj. kódu)
    - SF - upravený součet 5 driverů s hodnocením 0 – 5
    - Drivery exponentu:
        - návaznost na předchozí výsledky
        - flexibilita vývoje
        - rozhodnutí architektury/rizika
        - koheze týmu
        - vyspělost procesu (podle SEI CMM)
    - EM: multiplikátory úsilí (7 pro EDM, 17 PAM) – nové atributy

Funkční body

- normalizovaná metrika SW projektu
- měří aplikační oblast, nezkoumá technickou oblast
- měří aplikační funkce a data, neměří kód
- měří vstupy, výstupy, dotazy, vnitřní paměti, vnější paměti
- princip odhadu → velikost projektu x složitost x rizikové faktory
- jeden funkční bod = jedna cihlička
- u COCOMA byl problém odhadnout LOC

Funkční body vztažené k transakčním funkcím

- Externí vstupy (EI - External Inputs) → něco vstoupí do systému a uloží se do DB bez další odevzy (input, update, delete)
- Externí výstupy (EO - External Outputs) → něco vezmu z DB a zobrazím to, aniž by se DB změnila (select)
- Externí dotazy (EQ - External Enquiry) → vstup s následným zobrazením, něco se uloží a pak se to zobrazí

Funkční body vztažené k datovým funkcím

- Vnitřní logické soubory (ILF - Internal Logical Files) → mám data která spravuje moje aplikace
- Soubory vnějšího rozhraní (EIF - External Interface Files) → mám data která moje aplikace nespravuje ale přebírá je od někoho jiného. Něco co importujeme (každý den si stáhneme nové kurzy měn, které neuchováváme)

Neupravené funkční body (UFP)

- Před výpočtem musíme EI, EO, EQ, ILF, EIF roztřídit do skupin podle vah. Potom spočítame FP

FTRs	1-4 DETs	5-15 DETs	16+DETs
0-1	nízká	nízká	průměrná
2-3	nízká	průměrná	vysoká
4+	průměrná	vysoká	vysoká

• FTR = File Types (User Data Groups) Referenced
• DET = Data Element Type (Attribute)
• RET = Record Element Type (User View)

RETs	1-19 DETs	20-50 DETs	51+ DETs
1	nízká	nízká	průměrná
2-4	nízká	průměrná	vysoká
5+	průměrná	vysoká	vysoká

• FTR = File Types (User Data Groups) Referenced
• DET = Data Element Type (Attribute)
• RET = Record Element Type (User View)

Váhy:	nízká	průměrná	vysoká	celkem
EI	___ x 3 +	___ x 4 +	___ x 6 =	_____
EO	___ x 4 +	___ x 5 +	___ x 7 =	_____
EQ	___ x 3 +	___ x 4 +	___ x 6 =	_____
ILF	___ x 7 +	___ x 10 +	___ x 15 =	_____
EIF	___ x 5 +	___ x 7 +	___ x 10 =	_____

Neupravené funkční body celkem (UFP): _____ (součet)

Faktor technické složitosti (FTS)

- 14 charakteristik:
  - Jsou vyžadovány datové komunikace?
  - Je výkonnost kritická?
  - Požaduje systém on-line vstup dat?
  - Je kód navrhován s cílem znovupoužití?... (viz kniha)
- Každá charakteristika je hodnocená ve stupnici 0 – 5 takto:
  - 0 = bez vlivu
  - 1 = náhodný
  - 2 = mírný
  - 3 = průměrný
  - 4 = významný
  - 5 = podstatný

Výpočet

1. Identifikujte a spočtěte ILF, EIF, EI , EO, EQ. Pro každou ILF a EIF identifikujte počet RET a počet DET. Pro každou EI, EO a EQ, identifikujte počet FTR a DET
2. S použitím matice složitosti spočtěte váhy EI, EO, EQ, ILF, EIF (nízká, průměrná, vysoká).
3. Spočtěte Počet neupravených funkčních bodů.
4. Určete hodnoty 14 charakteristik systému.
5. Sečtěte hodnoty charakteristik – určíte tak Faktor technické složitosti (FTS) systému.
6. Určete Počet upravených FP systému.

$$FP = \left(0.65 + \left(0.01 \times FTS\ \right)\right) \times \left(UFP\right)$$

Závěr

- Otázky TIS I a TIS II
- Slidy vedení týmového projektu
- Slidy objektové metody návrhu IS
- Slidy ANANAS