Životní cyklus software
(dle unifikovaného procesu)

- Požadavky
- Analýza
- Design/návrh
- Implementace
- Testování

Každá fáze cyklu se dále probíhá v etapách:

- Zahájení
- Rozpracování
- Konstrukce
- Zavedení

Životní cyklus software
(jiný pohled)

- Nápad
- Neformální specifikace (odborný článek, úvodní studie)
- Formální specifikace (analýza)
- Dekompozice (návrh)
- Řešení komponent (modulů)
- Implementace komponent
- Testování komponent
- Integrace komponent do celku
- Testování celku (akceptační test)
- Instalace
- Provoz a údržba

Náročnost fází životního cyklu
Ze statistik pro velký systém o řádově stovky tisíc řádků kódu vyplývá, že úsíli věnované těmto fázím by mělo být rozděleno zhruba v poměru:

- Analýza 40%
- Návrh 40%
- Implementace 20%

Softwarové inženýrství

Proces

- Je vodítkem k vytvoření kvalitního sw produktu
- Poskytuje rámec pro řízení a organizování aktivit, které se snadno mohou vymknout kontrole
- Různé projekty vyžadují různé procesy
- Sw procesy jsou uzpůsobovány konkrétním potřebám
- Sw produkty (programy, dokumentace, data) jsou výsledkem aktivit definovaných SW procesem
- Dobrými indikátory funkčního sw procesu jdou kvalita, dodržení času a dlouhodobá realizovatelnost produktu.

Proč je proces důležitý?

- Umožnuje rozdělení práce
  - Každý člen týmu ví co má dělat
- Podporuje týmovou/individuální práci/komunikaci
  - Porozumění tomu, co se děje
- Usnadňuje řízení projektu
  - Manažéři vědí lépe, co se děje
- Umožnuje znovupoužití zkušenosti
  - Přenos mezi různými projekty
- Usnadňuje trénink
  - Lze ho standardizovat
- Podporuje zvyšování produktivity
  - Vývoj se může stát opakovatelným

Obecné (generické) fáze SW procesu

- Definice
  - Co?
    - Information enginnering
    - Software project planning
    - Requirements analysis
- Vývoj
  - Jak?
    - Software design
    - Code generatiob
    - Software testing
- Podpora a údržba
  - Změny
    - Corrective maintenance
    - Adaptive maintenance
    - Perfective maintenance
    - Preventative maintenance

Obecný procesní rámec (OPR)

- Ustanovuje základní obecný rámec pro procesy
- Definuje
  - Rámcové aktivity
  - Množiny úloh
    - Úkoly
    - Milníky
    - Kontrolní body
  - Zastřešující činnosti

Obecný procesní rámec: zastřešující činnosti

- Monitorování a kontrola
- Formální technické revize a kontroly (review)
- Zajišťování kvality SW
- Řízení a správa konfigurace
- Příprava dokumentů
- Management znovupoužití (Reusability management)
- Používání metrik
- Řízení rizik

Úroveň vyspělosti procesu

- CMM definuje jednotlivé úrovně vyspělosti SW
- Level 1 : Iniciální
  - Ad hoc sw proces, základní řízení projektu
- Level 2 : Opakovatelný
  - Schopný opakovat předchozí úspěchy, integrovaný proces řízení projektu
- Level 3 : Definovaný
  - Řídící a inženýrský procesy jsou zdukumentovány, standardizovány a integrovány s širšími organizačními procesy
- Level 4 : Řízený
  - K procesům a produktům je přistupováno pomocí detailních kvantitativních metrik a jsou pomocí nich měřeny
- Level 5: Optimalizující
  - Samozlepšujícíc se proces
  - Zpětná vazba

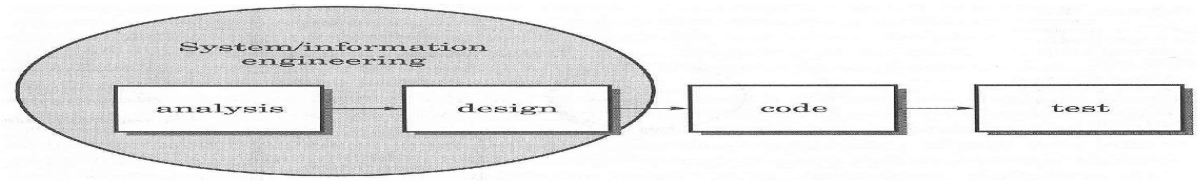
Modely životního cyklu SW

- Existují zavedené modely, např.
  - Lineární
  - Model vodopád

    - Spirálový model
    - Přírůstkový model
- Model životního cyklu určuje základní schéma postupu
- Životní cyklus by měl vždy začínat dostatečně přesnou specifikací a návrhem
- Není nutno realizovat celý systém najednou – naopak přírůstky poskytují uživateli dobrý pocit postupu práce

Lineární model (model vodopád)

- Nevrací se zpět
- Vhodný v případě jasného zadání
- Problémy:
    - Měření
    - Kontrola
    - skluzy



Prototypy

- v cyklech
- výsledkem fází jsou prototypy
- výhody
    - postup
    - měření
    - zpětná vazba
- problémy
    - zahození prototypu
    - poněkud vágní metody

Rapid Application Devolopement (RAD)

- krátký cyklus
- adaptace lineárního modelu
- týmy
- paralelní běh
- podprojekty

Evoluční (iterativní) modely

- řada modelů
- proces probíhá typicky v iteracích
- výsledek iterací bývají funkční produkty s omezenou funkcionalitou
    - tím se liší od lineárního i od prototypů

Metodika Rational Unified Process (RUP)

Základní činnosti	Fáze
1. požadavky	Zahájení
2. analýza	rozpracování
3. design	konstrukce
4. implementace	zavedení
5. testování	

Interativní vývoj v RUP

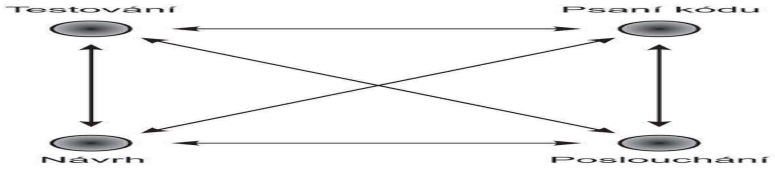
- objektivní posouzení stavu projektu
- rovnoměrnější pracovní vytížení vývojářského týmu
- testování meziverzí
- spolupráce s uživateli v průběhu celého projektu
- včasné rozpoznání nesrovnalostí mezi požadavky, návrhem a implementaci
- Snazší zapracování změn požadavků

4G metody (taky agilní metody)

- Metody 4. Generace
- Cílem je
    - Rychlé,
    - Automatizované,
    - Vysokoúrovňové
- Generování, vytváření produktů, dokumentů
- Nástroje
    - CASE systems
    - Database query language
    - Report generator
    - Screen painter
    - Charting/graphing tools
    - Spreadsheet
    - Application generator

Extrémní programování – XP (ne windows!!! – ani Vista)

- Pracuje se čtyřmi proměnnými
    - Kvalita
    - Čas
    - Náklady
    - Šíře zadání
- Hodnoty
    - Komunikace
    - Jednoduchost
    - Zpětná vazba
    - Odvaha
    - Respekt



Postupy XP

- plánovací hra
    - průzkum
    - závazek
    - řízení
- malá verze
- metafora (příběh vývoje)
- jednoduchý návrh
- testování
- refaktorizace
- párové programování
- společné vlastnictví
- nepřetržitá integrace
- 40-ti hodinový pracovní týden
- Zákazník na pracovišti
- Standardy pro psaní zdrojového kódu

Řízení projektů

Projekty obecně

Vznik SW dílá je typicky ralizován jako 1 nebo více projektů

Projekt je dobře definovaná posloupnost činností, která má určen začátek a konec, je zaměřena na dosažení určitého cíle a je uskutečňována pomocí zdrojů – lidí, věcí, nástroje a prostředků.

Řízení projektů

- Řízení projektů zahrnuje:
  - Plánování
  - Monitorování
  - Kontrolu
    - Lidí, procesů, událostí
  - Každý něco řídí, rozsah závisí na konkrétní roli v rámci projektu
- Každý něco řidí, rozsah závisí na konkrétní roli v rámci projektu
- SW musí vznikat řízeně
  - Příliš komplexní
  - Dlouhodobá činnost

4P

- Úspěšní manažeři se musí zaměřit na tzv. 4P:
  - Personál (people)
  - Produkt
  - Proces
  - Projekt
- Plán projektu:
  - Dokument definující 4P s cíle zajistit finanční a časovou efektivitu

Lidé – typické úlohy

- Nábor, výběr, výkonnost, trénink, vzdělání, školení, odměňování, kariérní růst/vývoj, organizace, práce v týmu, vývoj.

Lidé a hráči

- Senior manažeři
  - Klíčové obchodní vize a myšlenky
- Projektový manažeři
  - Plány, motivace, organizace a kontrola „praktiků“
- Praktici/inženýři
  - Znalosti a dovednosti
- Zákazníci
  - Požadavky
- Koncoví uživatelé
  - Interakce, zpětná vazby

Vedoucí týmu

- Intenzivní a náročná činnost
- Vyžaduje člověka
  - Zkušeného
  - Talentovaného
  - Odborně zdatného
  - Vhodný temperament a povahový tup
- Zkušení praktici nebývají často dobrými vedoucími projektu
- Není dobré techniky a programátory nutit do role vedoucího týmu

Úlohy vedoucího týmu (J. Weinberg)

- Motivace
  - Schopnost ostatbí motivovat k tomu, aby pracovali, jak nejlépe umí
- Organizování
  - Práce s procesy, plánování, kontrola
- Myšlenky a inovace
  - Schopnost motivovat spolupracovníky ke kreativitě a i v prostředí s danými hranicemi a omezeními

Dovednosti dobrého vedoucího týmu

- Schopnost řešit problémy
  - Identifikace, diagnóza, nacházení relevantních a přiměřených řešení

- Identita manažera
    - Na strunu jednu je třeba čídit a vést
    - Na stranu druhou důvěřovat lidem a nechat jim dostatek autonomie
- Ocenění úspěchů
    - Jednak ocenění dílčích úspěchů členů
    - Netrestat podstupování kontrolovaného rizika
- Týmová komunikace
    - Verbální i neverbální
- Sebeovládání

Tým

- Neexistuje optimální organizace týmu
- Závisí na okolnostech, projektu, lidech
- Typy
    - Demokratický decentralizovaný
    - Kontrolovaný decentralizovaný
    - Kontrolovaný centralizovaný
- Dobře fungující tým
    - Vzájemná důvěra mezi členy
    - Přiměřené rozložení dovedností
    - Konfliktní typy mohou být vyloučeny z týmu
- Manažer musí na začátku jasně stanovit odpovědnost a role týmu

Selský rozum radí k projektům

- Začněte „pravou nohou“
    - Pracuj tvrdě a smysluplně
- Udržuj vysoukou hybnost (rychlost)
    - Tempo má tendenci klesat
- Měř postup
- Dělej rozumná rozhodnutí
    - KISS (keep is simple and stupid)
    - Používej již hotové
- Pouč se z chyb
    - Dělej zpětnou analýzu

W5HH princip

12 Essential Attributes for Successful software

1. effective project planning
2. effective project cost estimating
3. effective project measurements
4. effective project milestone tracking
5. effective project quality control
6. effective project change management
7. effective development processes
8. effective communications
9. capable project managers
10. capable technical personnel
11. significant use of specialists
12. substantial volumes of reusable materiál

Proces vývoje SW Objektově orientované modelování v UML

Co je to UML

- Unified Modeling Language
- Obecný jazyk (notace) určený k modelování systémů
- Zahrnuje současné nejlepší postupy (best practices) v modelování (vychází z řady předchozích vizuálních modelovacích technik)
- Je neutrální vůču metodologiím a jazykům
- Průmyslový standard pro modelování OO systémů

- Standardizační orgán: OMG (Object Managemend Group)
- Základní předpoklad OO modelování
    - SW může být modelován jako soubor spolupracujících objektů

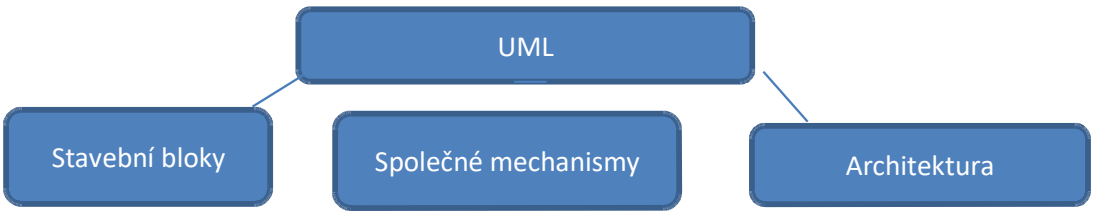
Proč unifikovaný

- Snaží se o unifikaci různých domén
- Vývojový cyklus
- Aplikační doména
    - Realtimové systémy, distribuované systémy
- Implementační jazyky
    - C++, Java, #
    - Je nezávislý na jazyce
- Vývojové procesy
    - Např. Unified Process (UP), Rational Unified Process (RUP)
- Vlastní interní pojmy
    - Vnitřní jednota a konzistence

Oblasti pokrývaní UML

- Funkční model
    - Diagramy popisující fungování systému
- Objektový model
    - Popis struktury systému pomocí tříd, objektů, atributů, operací, relací, vazeb, asociací
- Dynamický model
    - Popisuje dynamické aspekty systému
    - Interakce, chování systém v čase, spolupráce tříd, sekvence volání, stavy systému a přechody mezi nimi

Struktura jazyka UML



- Stavební bloky
    - Předměty
    - Vztahy
    - Diagramy
- Společné mechanismy
    - Specifikace
    - Ozdoby
    - Podskupiny
    - Mechanismy rozšiřitelnosti
- Architektura
    - Pohled případů užití
    - Logický pohled
    - Procesní pohled
    - Implementační pohled
    - Pohled nasazení

Struktura jazyka UML II

- **Stavební bloky** - základní modelovací elementy
    - **Předměty**: modelovací elementy – třídy, rozhraní, případy užití, interakce, balíčky, poznámky
    - **Vztahy** (relace): specifikují sémantickou souvislost 2 a více předmětů
    - **Diagramy**: pohled na UML modely: zobrazující soubory předmětů vizualizující CP resp. Jak bude systém řešen

- **Společné mechanismy –** společné způsoby dosahování specifických cílů
    - **Specifikace**: textové popisy sémantiky modelovaných elementů
    - **Ozdoby**: doplňující vizuální detaily modelovaných elementů
    - **Podskupiny** (common division): specifické způsoby přemýšlení o světě (vidění/modelovíní světa) – např. vztahy klasifikátor – instance, rozhranní-implementace
    - **Mechanismy rozšiřitelnosti**: specifikuje mechanismy pro rozšíření modelovacích elementů jazyka pro specifické účely (omezení, stereotypy, označené hodnoty)
- **Architektura** – zachycení **strategických aspektů struktury** systému jako celku pomocí UML:
    - **Pohled případu užití**: zachycuje základní požadavky na systém a poskytuje východisko pro tvorbu dalších pohledů
    - **Logický pohled**: zachycuje slovník problémové oblasti (domény) pomocí sady tříd a objektů a jejich vztahů
    - **Procesní pohled**: modeluje vykonavatelná vlákna a procesy v systému
    - **Implementační pohled**: modeluje soubory a komponenty, které utvářejí kódový základ systému
    - **Pohled nasazení**: modeluje fyzické nasazení komponent na výpočetní uzly jako jsou počítač a jiné HW zařízení

Modelování

- SYSTÉM organizovaná množina komunikujících (propojených) součástí
    - Přirozený X umělý systém
    - Podsystém
- MODELOVÁNÍ - PROSTŘEDKY PRO ZVLÁDNUTÍ KOMPLEXNÍHO A SLOŽITÝCH SYSTÉMŮ
    - Vytváření abstrakce systému
    - Zaměření na důležité aspekty systému
    - Model systému je sada všech modelů vytvořených během vývoje
- POHLED – zaměřuje se na část modelu a zobrazuje ji tak, aby byla jasní a srozumitelná

Abstrakce

Proces výběru některých aspektů a charakteristik systému a vyloučení jiných irelevantních

- Abstrakce je vždy dělána s účelem
- Je možných mnoho různých abstrakcí
- Všechny jsou neúplným popisem reality
- **Nepotřebujeme úplnost, ale adekvátnost modelu vzhledem k jeho účelu!**

Typy abstrakce

- **Klasifikace-** seskupuje podobné instance objektů
    - **Najdeme společné vlastnosti a ignorujeme jedinečné**
- **Agregace –** seskupení různých objektů objektů
    - **Ignorujeme odlišnosti a zaměříme se na to, že dohromoday utvářejí celek**
- **Generalizace –** seskupení podobných množin objektů
    - **Rozdíl mezi klasifikací a generalizací**
    - **Klasifikace –** aplikuje na individuální instance objektů
    - **Generalizace –** aplikuje se na množiny objektů (třídy)

Datové typy, abstraktní DT a instance

- **Datový typ:**
    - Abstrakce v kontextu programovacího jazyka
    - Jednoznačné jméno
    - Vymezuje množinu přípustných hodnot (členů, instancí)
    - Definuje množinu přípustných operací
    - P5.:
        - Integer
        - String
        - Zamestnanec
- **Abstraktní datový typ**
    - DT definovaný pomocí implementačně nezávislé specifikace
    - Např. matematické pojmy

        - Množiny, mapa, sekvence
    - Nemůže mít instance

Třídy a objekty

- **Třída**:
    - abstrakce v OO modelování zachycuje koncept, jeho hranice, význam a operace
    - Prostředky jazyka pro definici nových datových typů
    - Explicitně propojuje datové struktury (viz. Např. C a stuct) s operacemi na nich (funkce, metody)
    - Třída definuje
        - Operace
        - Atributy
- **Objekty**
    - Je instancí třídy
    - Např. třída Zamestnanec a její konkrétní instance Karel Novák, instlalatér.

Proč jsou třídy užitečné?

- Užitečná prostředek konceptualizace světa a jeho modelování
    - Neboli nazýváme věci jejich jmény (např. typy kočka, pes, zvíře…)
    - Spojení operací s odpovídajícími daty
    - Vhodná základ pro modularizaci kódu
- Snižuje složitost problému

- Dědičnost (inheritance)
- Zapouzdření (encapsulation)
- Polymorfismus (pomymorphism)

OO modelování

- **Aplikační doména**
    - Reprezentuje všechny aspekty z uživatelského pohledu
    - Např. procesy, účastníky, uživaele,…
    - Pozor: v čase se vyvíjí
- **Domény řešení**
    - Modelujem prostor všech možných systémů řešících problém
    - Návrh objektů, tříd, procesů atp.
    - Bohatší na AD

- Objektově orientovaná analýza
    - Zabývá se modelováním aplikační domény
- Objektově orientovaný design
    - Modelování démy řešení

Diagram případů užití (Use Case diagram)

- Zachycuje chování systému z pohledu uživatele
- Definuje hranice systému, uživatele systému a jednotlivé případy užití
- Vyvíjen společně s modelem tříd
- Pomáhá:
    - Zaznamenat data a funkční požadavky
    - Plánování iterací vývoje
    - Kontrola a validace systému
- Dynamický model začíná analýzou use-case
    - Řídí celý vývojový proces
    - Veškeré požadovaná funkcionalita je popsána v use-casech
        - Use-case model je vyvíjen inkrementálně

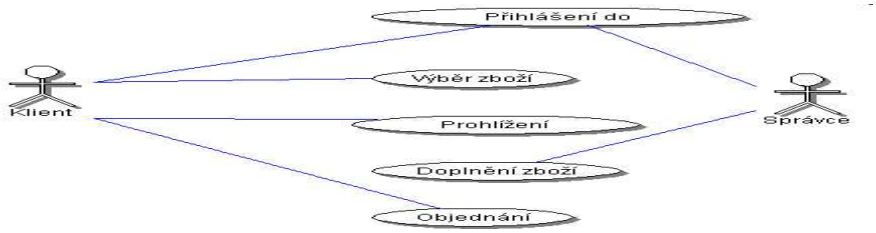
Use case – uživatel (actor)

- Externí entity, která interaguje ses sytémem
- Může to být člověk nebo jiný systém
- Poskytuje vstupy a přijímá výstupy systému

- Spíš než konkrétního uživatele se jedná o role
    - Více rolí na 1 uživatele;
    - Více uživatelů v 1 roli
- Uživatel je v podstatě třídou (klasifikátor)
    - Konkrétní uživatel pak jeho instancí

Případ užití (PU)

- Část systému realizující jeho určitou specifickou funkcionalitu z hlediska uživatele
- Specifikuje
    - Interakce s uživatelem
    - Sekvence akcí/událostí iniciovaných uživatelem
    - Ignorujeme zatím výjimky chybové stavy
    - Dále volitelně též
        - Vstupní a výstpuní podmínky
        - Jiné kvalitativní požadavky
    - Shrnuje v sobě více možných scénářů řešící tentýž případ užití



Specifikace případu užití

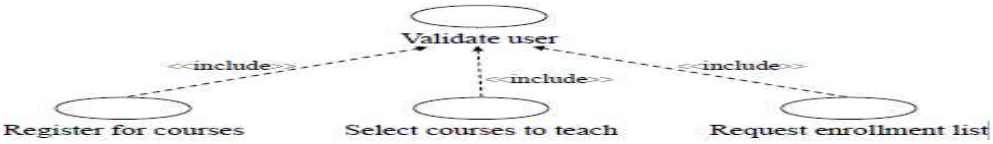
- Textová specifikace jednotlivých případů užití
- Není přesně definováno v UML
- Má možná větší významna než samotný diagram
- Specifikuje
    - Název případu užití (PU)
    - Účastnící (iniciátor, komunikující učastník)
    - Tok událostí (co se bude dít během PU)
        - Má podobu číslovaného seznamu událostí
        - Potřebná data
        - Interakce s uživatelem
    - Vstupní podmínka (např. uživatel je přihlášen)
    - Výstupní podmínka (uživatel zaplatil)
    - Kvalitativní požadavky (např. systém odpoví do 30 vteřin)

Scénaře

- PU je poměrně abstraktní specifikace která může být realizována různými scénáři
    - Např. výběr zboží
        - Na základě doporučení
        - Vyhledávání
        - Přehled kategorií
    - Obsah scénáře
        - Název
        - Účastníci
        - Tok událostí

Use case Model Detaily – Includy

- Více PU může sdílet tutéž funkcionalitu- použijeme
- Rozdělíme use-casy a pak je znovu- používáme



Use case model detaily – Extend

- Umožňuje rozšířit chování PU pomocí jiného případu užití
- Vhodné např. pro ošetření výjimečných stavů

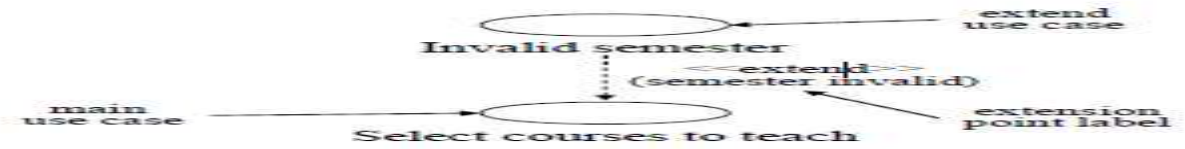
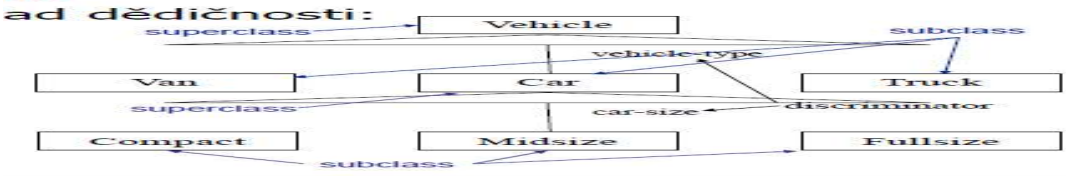


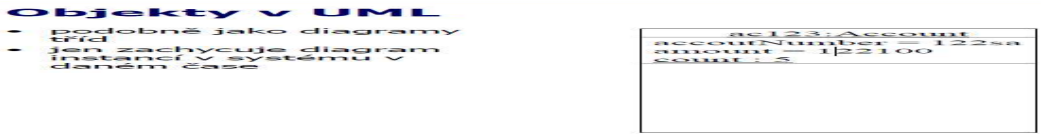
Diagram tříd

- Popisuje strukturu systému pomocí tříd a jejich vztahů
- Příklad dědičnosti:

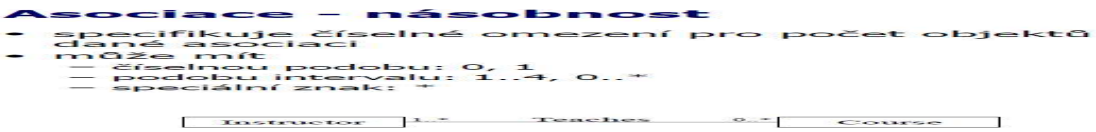


Třídy v UML notaci

Objekty v UML



Asociace



Sekvenční diagram

- Reprezentuje účastníky a objekty a jejich interakce v čase
- **Prvky diagramu**
  - Objekty – znázorněné obvykle jako sloupce
  - Interakce mezi objekty (zprávy) – orientované šipky mezi objekty
  - Události – události, které vyvolaly interakci
  - Reakce – odezvy na události (výstupy)

- Časová osa – pro vyznačení sledu událostí

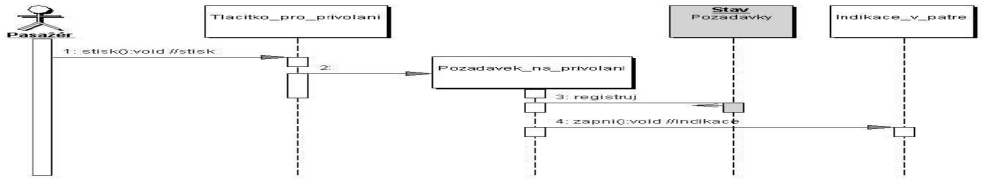
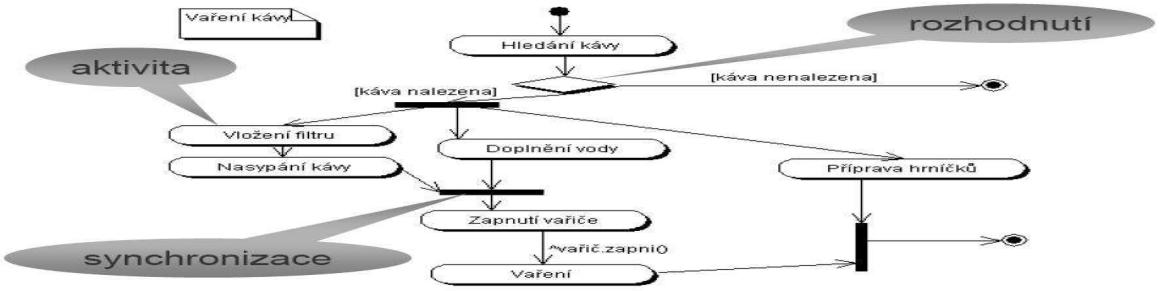


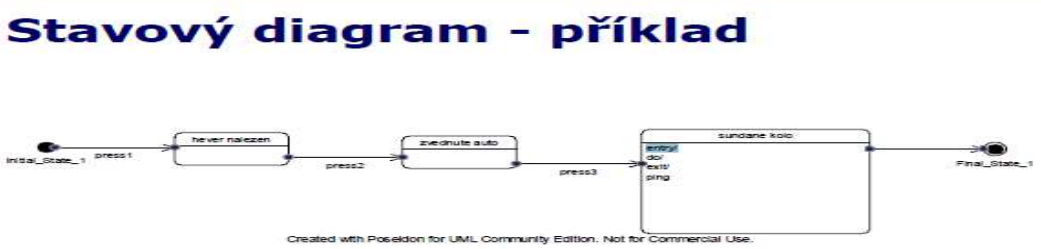
Diagram aktivit

- Používá se pro dokumentaci případu užití (workflow)
- Popisuje chování systému prostřednictvím aktivit
- Aktivitua je modelovací element reprezentující sadu operací
- Je podobný diagramu toků (ten v UML není)
- **Prvky diagramu**
  - **Počáteční a koncový stav** – tlustá tečka (v kolečku pro konc.stav)
  - **Aktivity** – elipsy
  - **Přechody mezi aktivitami** – šipky
  - **Synchronizační body** – tlusté obdélníky
  - **Rozhodnutí** – kosočtverec



Stavový diagram

- Popisuje dynamiku chování individuálního objektu (nikoliv skupiny objektů)
- Primitivy jsou
  - Stavy a
  - Přechody mezi stavy
- Stav reprezentuje určitou množinu hodnot daného objektu (stav objektu)
- Je vhodný zejména pro popis složitějšího chování objektů, kde chceme přesně specifikovat toto chování



Získávání a zpracování požadavků

Sběr a zpracování požadavků – aktivity dle unified Process

- Modelování domény
- Modelování případu užití
- Specifikace uživatelského rozhraní a modelů
- Validace požadavků

Sběr požadavků

Požadavek

vlastnosti, kterou musí systém mít, nebo omezení které musí splňovat, aby byl akceptován zákazníkem

Zachycování požadavků

Vyžaduje spolupráci různých skupin účastníků => DISPROPORCE ZNALOSTÍ

Výzva: Jak překonat disproporci

Zahrnuje

- Identifikace
  - Uživatelů scénaře
  - Případu užití
  - Upřesnění PU
  - Vztahů mezi PU a uživateli
  - Analytických tříd
  - Nefunkčních požadavků

Proč je zpracování požadavků těžké?

Cílem je vytvořit správný systém

Systém splňující požadavky uživatelů

Ale, uživatelé často neví, co potřebují

- Mnoho kategorií uživatelů (znají jen své potřeby a doménu)
- Nevidí „ucelený“ obraz systému
- Nemusí vědět, které aspekty jejich práce lze zpracovat počítačově

Z hlediska softwarového inženýra se při zpracování požadavků jedná o často p
proces „objevování a učení“

- Je třeba uživatelům vysvětlit, co jsme zjistili aby mohli schválit, že to splňuje jejich potřeby
  - Uživatelé musí rozumět naší specifikaci
- Leč, uživatel není SW specialistou

Proces zpracování požadavků – přehled

- **Identifikace a pochopení uživatelských potřeb**
  - Definice cílů systému
  - Vytvořit kandidáty požadavků
  - Definice systémových omezení
  - Definice hranic systému
- **Zjištění proveditelnosti**
  - Ekonomická
  - Technická
  - Operační
  - Organizační dopady
- **Pochopení, zachycení a dokumentace systémových požadavků**
  - Statický pohled (persistentní data
    (domain model + specifikace)
  - Dynamický (procesy + omezení)
    případy užití + specifikace)
- **Validace požadavků**
  - Kritéria pro přijetí kompletního systému uživateli (acceptance tests)

Uživatelské potřeby – identifikace

- Sběr dat v aplikační doméně
  - Prozkoumání – existující dokumentace
  - Pozorování – pracovní činnosti
  - Rozhovory – dotazníky, osobní
  - Prototypování – rozhraní, funkce
    - Zaměřujeme se na problémy, ne na řešení
    - Rozlišovat potřeb od přání+ hodnotit důležitost potřeb
- Zpřesnění cílů systému, seznam požadavků, seznam omezení, hranice systému, atd.

- Návrh rozsah projektu (co je zahrnuto, co nikoliv)
- Požadavky na specifikaci -→ specifikace systémových požadavků (RAD – Requirements Analysis Document)
- **SLOUŽÍ JAKO KONTRAKT MEZI ŘEŠITELI A UŽIVATELI!**

Určení proveditelnosti

- Ekonomická
  - Náklady (vývoj, provozu versus příjmy
- Technická
  - Dostupnost technologií
  - Riziko nových technologií
- Operativní
  - Dostupnost sil k provozování systému
  - Úprava pracovník praktit (redesing)
- Organizační
  - Politika, školení a trénink, přijetí uživateli
- Právní
  - Ručení a odpovědnost: copyright, patenty

Zachycení a dokumentace systémových požadavků

- Statické požadavky (persistentní data) – domain model
  koncepty a třídy aplikační domény
  - Vznik slovník pojmů (datový slovník) umožňující komunikaci participantů
- Dynamické požadavky (zpracování + omezení) → use.case model
  Funkční požadavky – popisuje interakce systému s okolím a jeho funkcionalitu (nezávisle na implementaci)
  Nefunkční požadavky – požadavky, které nejsou přímo spojeny s funkcionalitou systému, ale přesto ovlivňují jeho chování z pohledu uživatele
  Např. doba odezvy, kapacita, bezpečnost, atd.
  Pseudo požadavky – omezení implementace daná zadavatelem
  Např. implementační jazyk, platforma, atd.

SPECIFIKUJE JEN
CO ne, JAK to bude

Artefakty

- Model domény – třídy a jejich asociace zachycuje datové požadavky
- Use case model – zachycuje funkční požadavky
- Uživatelé
- Slovník
- Specifikace domény – popis tříd a asociací
- Specifikace případů užití + scénář
- Už rozehranní a prototyp
- Popis architektury

Identifikace tříd

- Třídami mohou být
  - Obchodní entity (např., objednávky, účty,..)
  - Reální objekty (zákazník,auto,..)
  - Události (rezervace, …)
- Přirozeně lze s pojmů apl. Domény
  - Classes: podstatná jména(jmenné fráze
  - Asociace: slovesa/ slovesné vazby
- Vše v jednotném čísle/aktivní formě

Máme správné třídy?

- Jsou některé třídy redundantní?
- Nejsou některé třídy irelevantní ve vztahu k aplikační doméně?
- Nejsou některé třídy vágně/chybně definované?
- Měly by některé třídy skutečně být ?
- Nepopisují některé třídy operace?
- Nepopisují některé třídy role?
- Nepopisují některé třídy implementační konstrukty?

Identifikování uživatelů

Kdo a jak užívá systém
Jaké role X skupiny se vyskytují
Kdo zadává a poskytuje informace systému
Kdo instaluje, zapíná a vypíná, spravuje systém
Jaké jiné systémy interagují
Stane se něco v čase

Vstupní zařízení nejsou uživateli

Čas

Stručně popsat roli, v níž vystupuje.

Identifikace scénářů

Stejné otázky jako i Use casů

- Typy
  - As-is
  - Vize
  - Evaluace a testování
  - Trénink

2 přístupy k analýze

1. Shora-dolů — od PU ke scénářů
2. Zdola-nahoru — opačně

Scénáře často pro popis chyb a alternativních toků.

Identifikace případů užití

Úlohy vykonávané uživatelem
Jak zpracována actor informace (create, store, change, remove, or read)?
Externí změny
Události o kterých má být uživatel informován
Podpora a správa

Názvy: přítomní čas, slovesné fráze in činný rod
V popisu užívat terminologie

Extend
Include X extend

Nefunkční požadavky

- výkon
- spolehlivost a robustnost
- operační prostředí
- usability
- podpora (přízpůsobování, spravovatelnost, přenositelnost)
- životní cyklus — plán, rozpočet…
- implementace
- interface
- balíčky
- právní

Validace

- **koreknost**
- **úplnost**
- **konzistence**
- **jednoznačnost**
- **reálnost**
- Validační testy + akceptační protokol

Systémová analýza

Cíl: strukturovat model případů užití do formy, která je robustní a spravovatelná během životního cyklu SW.

- Předpokládáme ideální implementační prostředí
  - Nezohledňujeme: hardware, DBMS, programovací jazyk, atd. to se možná bude měnit
- Specifikujeme všechny logické třídy v systému, včetně asociací a případně seskupení do balíčků (packages)
- Distribuujeme chování use.case modelu do tříd analytického modelu
  - Explicitně specifikujeme, která třída je odpovědná za dané chování use-casu

Proč systémová analýza

Proč okamžitě neimplementujeme?

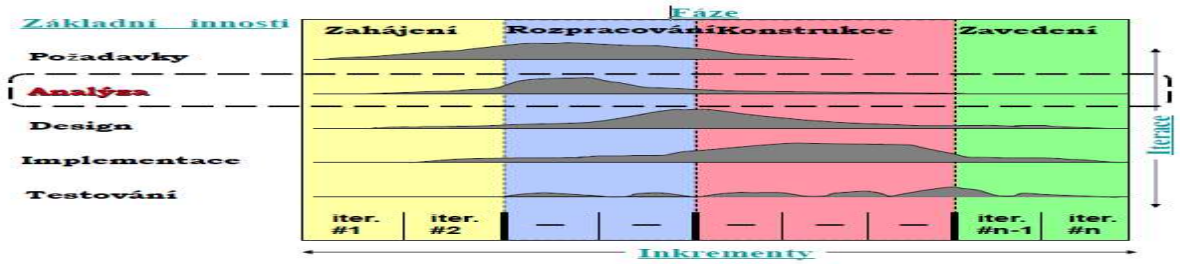
Výhody analytického modelu:

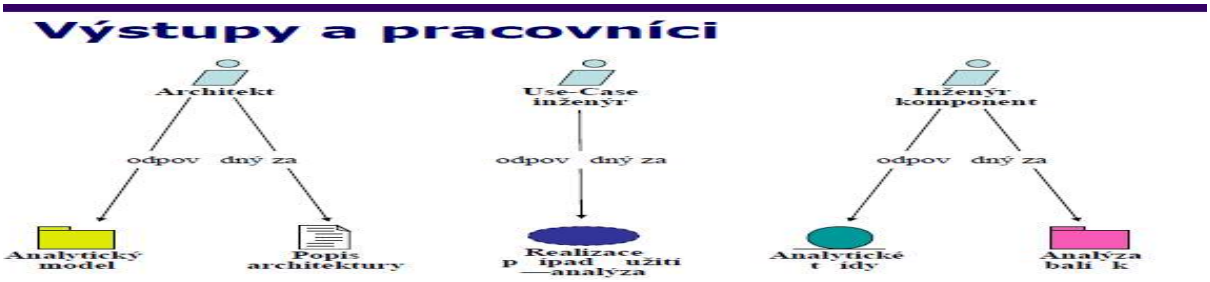
- Přesnější a úplnější specifikace požadavků – odstranění nejednoznační a duplicit
- Popsána v jazyce vývojáře – formálnější/přesnější
- Strukturuje požadavky pro lepší porozumění a udržování
- První přibližní k návrhu

Co dělat s analytickým modelem?

- Stále udržovat aktuální
- Po ukončení analýzy zahodit
- Ani jedna cesta není optimální

USE-CASE MODEL	ANALYTICKÝ MODEL
• Jazyk zákazníka • Externí pohled • PU strukturují externí pohled • Kontaktn mezi zákazníkem a vývojářem • Může obsahovat nejednoznačnosti, duplicity, mezery • Zachycuje funkcionalitu systémy	• Jazyk vývojáře • Vnitřní pohled na systém • Analytické třídy strukturují vnitřní pohled • Užíván vývojáři k pochopení systému • Neobsahuje nejednoznačnosti, duplicity, mezery, … • Náčrt realizace systému





Výstupy

- **Analytický model** – konceptuální strukturovaný model upřesňující požadavky a strukturuje je
- **Popis architektury** – globální pohled na systém zohledňující z hlediska architektury důležité prvky
- **Realizace PU-analýza** – mapování PU na třídy a jejich interakce
- **Analytické třídy** – realizují funkční požadavky, jsou 3 typů:
  boundary (hraniční), control (kontrolní/řídící) a entity (entity)
- **Analýza balíčků** – organizace tříd do menších skupiny které se lépe spravují

Pracovníci

- **Architekt** – architektura, její popis, integrita analytického modelu; zajišťuje korektnost, konzistenci, srozumitelnost

- **Use-case inženýr** – integrita jedné nebo více realizace případů užití, ručí za to , že budou odpovídat požadavkům
- **Inženýr komponenty** – definuje a spravuje odpovědnosti, atributy, vztahy a speciální požadavky jedné nebo více analytických tříd

UP – aktivity

• Analýza architektury
– identifikuje: analytické balíčky; evedentní anal. třídy; společné speciální požadavky
• Analýza případů užití
– identifikace anal. tříd
– distribuce chování do anal. tříd
– specifikuje interakce mezi AT
– zachycení speciálních (nefunkčních) požadavků
• Analýza tříd
– identifikuje hlavní odpovědnosti tříd (operace)
– …. atributy, vztahy…
– popis netriviálního chování
– zachycení speciálních (nefunkčních) požadavků
• Analýza balíčků
– struktura balíčků a vazeb mezi nimi

Činnosti analýzy: od případů užití k objektům

Identifikace analytických objektů
– entity objects
– boundary objects
– control objects
• Mapování objektů na PU pomocí sekvenčních diagramů
• Modelování interakcí pomocí CRC žtítků
• Identifikace
– asociací
– agregací
– atributů
• Modelování chování individuálních objektů
• Modelování vztahů dědičnosti
• Revize a kontrola analytického modelu

Heuristiky pro mapování jazyka na komponenty modelu (dle Abotta)

Prvek jazyka	Modelová komponenta	Příklad
Konkrétní podstatné jméno	Instance	Alice
Obecné podstatné jméno	Třída	Policista
Sloveso „činnosti"	Operace	Vytváří, dělá, vybírá
Sloveso „být"	Dědičnost	Je druhu…, je jedním z…
Sloveso „mít"	Agregace	Má, skládá se z…, zahrnuje…
Způsobové sloveso	Omezení	Musí být…
Přídavné jméno, adjektivum	Atribut	Popis nehody

Analytické třídy

Abstrakce jedné nebo více finálních tříd implementujících systém
• popisy jsou konceptuální, nikoliv implementační
– atributy: konceptuální, ne konkrétní jazyk
– chování: definováno textovými popisy odpovědností
– vztahy: konceptuální
• třídy jsou:
– boundary – entity – control
(návrh obsahuje obyčejně mnohem víc tříd — až 5x…)
zaměřují se jen na funkční požadavky

Principy dělení případů užití

Každý use-case rozdělen až na úroveň:
– **1. boundary tříd** · funkcionality přímo související s okolím systému
– **2. entity tříd** · ukládání a správa dat a informací

– **3. control třídy** · propojení mezi 1. a 2., tok dat, řízení atd.

Cíl: lokalizace změn tak aby výsledkem byl stabilní systém

V praxi mnoho rozhodnutí, kam jaké funkce umístit

Analýza tříd

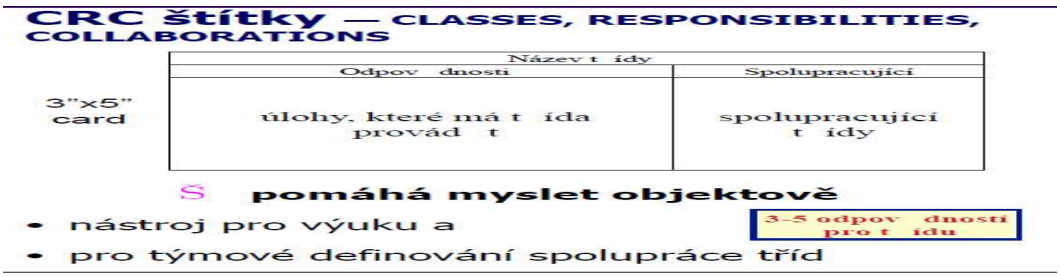
- **Identifikace odpovědností**
  – z rolí v PU

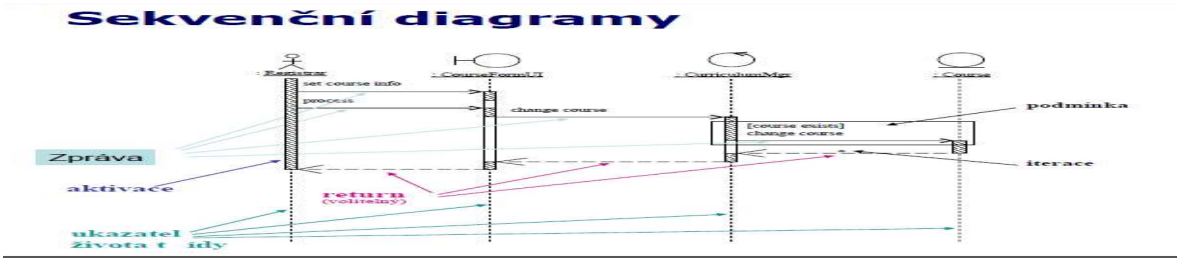
zvažuj: diagramy tříd, sekvenční, toky událostí

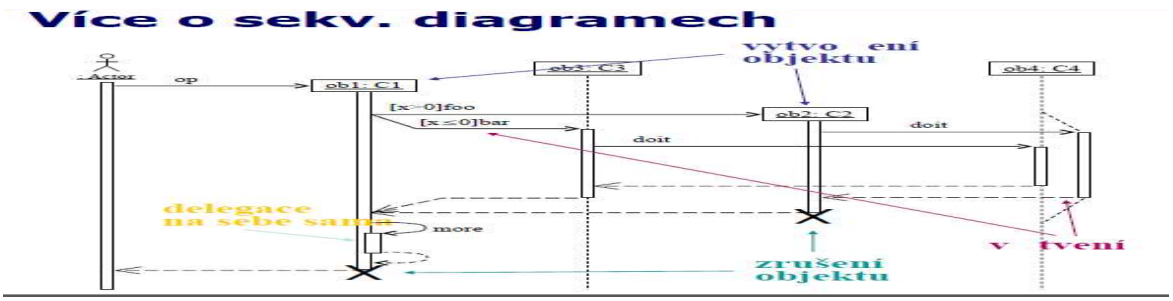
- **Identifikace atributů**
  – boundary třídy
  – entity třídy
  – kontrolní třídy ·
  jen zřídka mají atributy

• **Identifikace asociací, agregací, generalizací**
  – ze vztahů, zpráv, sekvenčních diagramů
  – ze sdílených chování mezi analytickými třídami

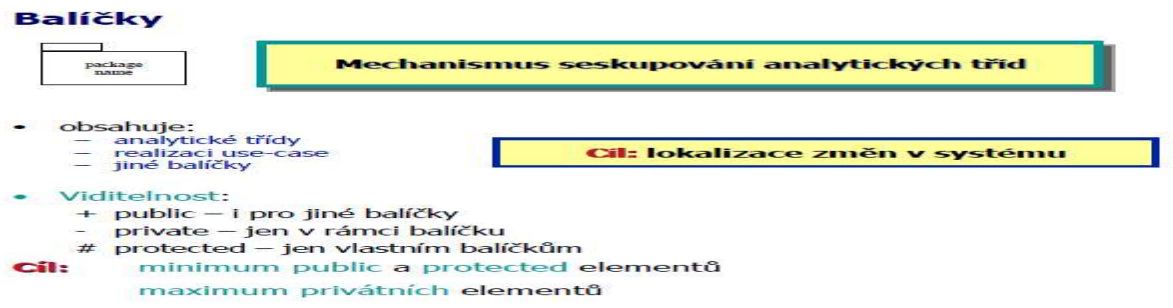
CRC štítky — CLASSES, RESPONSIBILITIES,COLLABORATIONS

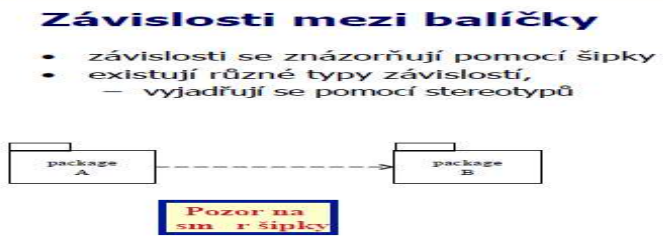


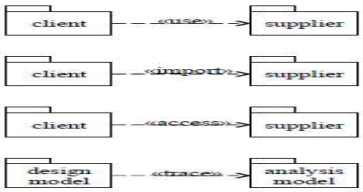




Balíčky







Revize analýzy

- Je vhodné na konci analýzy provést
- Kritéria
  - Korektnost
  - Úplnost
  - Konzistence

Návrh systému

Návrh – úvod

Přizpůsobování logické struktury analytického modelu na implementační prostředí a příprava na implementace.

- Zvaž vliv **nefunkčního požadavků**
- Zahrň **globální požadavky na systém**
- Zvaž **implementační prostředí**
- **Plně specifikuj každou třídu** včetně všech vlastností a operací
- **Rozděl implementační práce** do zvládnutelných celků → subsystémy
- Specifikuj hlavní rozhraní (interface) mezi subsystémy.

Kdy se začít věnovat systémovému designu?

- Až se ustálí analytický model a je požadováno jen minimum změn

Analytický model

- Konceptuální model
- Design → obecný
- Méně formální
- Levnější vývoj
- Méně vrstev
- Zaměření na interakce
- Nástin návrhu
- Nemusí být nutně dále udržován, spravován

Model návrhu

- Fyzický model
- Implementace → specifický
- Více formální
- Dražší vývoj
- Více vrstev
- Zaměření na sekvence
- Implementace návrhu
- Udržován a spravován v rámci životního cyklu

Cíle návrhu

Nač se zaměřit (vlastnosti) a co optimalizovat?

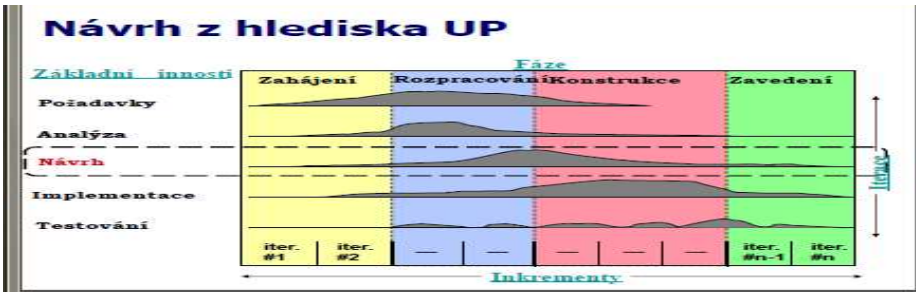
- Vlastnosti odvozeny hlavně s nefunkčních požadavků
- Vybrané vlastnosti (cíle) ovlivňují rozhodnutí učiněná během návrhu (zejména jeli třeba dělat kompromisy).
- **Obyčejně lze najednou zahrnout jen malou podmnožinu nefunkčních požadavků (jsou často v rozporu)**
- **Potřeba priorit dílů návrhu**
- **Příklad**: prostor vs. Rychlost; doba vývoje vs. Kvalita, kvalita vs. Cena

Aktivity návrhu

- Identifikace cílů návrhu
  - Identifikace

    - Přiřazení priorit
    - Dle cílů se optimalizuje
- Návrh základní dekompozice na podsystém
    - Rozklad na podsystémy
    - Na základě případů užití a analytického modelu
- Zpřesnění návrh podsystémů
    - Musí se zaměřovat na podporování cílů návrhu

Návrh z hlediska UP



Výstupy a pracovníci

Výstupy

- **Model návrhu** – popis fyzické realizace PU; zaměření na funkční i nef. Požadavky, společně s omezením impl. Prostředí
- **Model nasazení** – popis fyzické distribuce systému na jednotlivý HW uzly
- **Popis architektury** – 2 pohledy na arch. – **návrh** a **nasazení.**
- **Realizace PU** – návrh – popis realizace PU v pojmech návrhových tříd a objektů
- **Návrh tříd** – abstrakce tříd ve vztahu k implementaci systému
    - **Již v daném programovacím jazyce.**
- **Návrh subsystémů** – distribuce komponent da částí
- **Interface** – specifikace operací tříd a subsystémů
    - **Oddělení rozhraní od implementace**

Identifikace dílů – příklady cílů dle různých kritérií

Výkon-

- **Doba odezvy**
- **Kapacita**
- **Paměť**

Spolehlivost-

- **Robustnost**
- **Spolehlivost**
- **Dostupnost**
- **Odolnost vůči chybám**
- **Zabezpečení**
- **Bezpečnost (neohrozeni)**

Uživatelské cíle

- **Užitek**
- **Použitelnost**

Údržba a spravovatelnost

- **Rozšířitelnost**

- **Modifikovatelnost**
- **Adaptabilita**
- **Portabilita**
- **Srozumitelnost**

Náklady

- **Vývoje**
- **Zavádění**
- **Upgrady**
- **Údržba**
- **Správa**

Návrh subsystémů

Název subsystému	**Subsystém slouží j rozdělení komponent návrhového modelu do lépe zvládnutelných částí**

- Může obsahovat:
  - Návrhové třídy
  - Jiné subsystémy
  - Realizace use-casů
  - Interface
- **Use case, lze navrhnout jako spolupráci subsystémů**
  - **Třídový diagram subsystémů**
- **Lze je využít v diagramech spolupráce**
  - **Umožňuje hierarchickou dekompozici**
- **Mohou být zpřesněním balíčků nebo přímo součástí návrhového workflow**

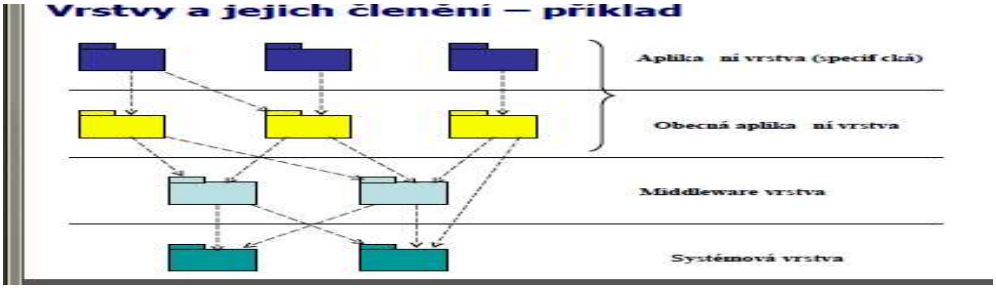
Heuristicky pro seskupování do subsystémů

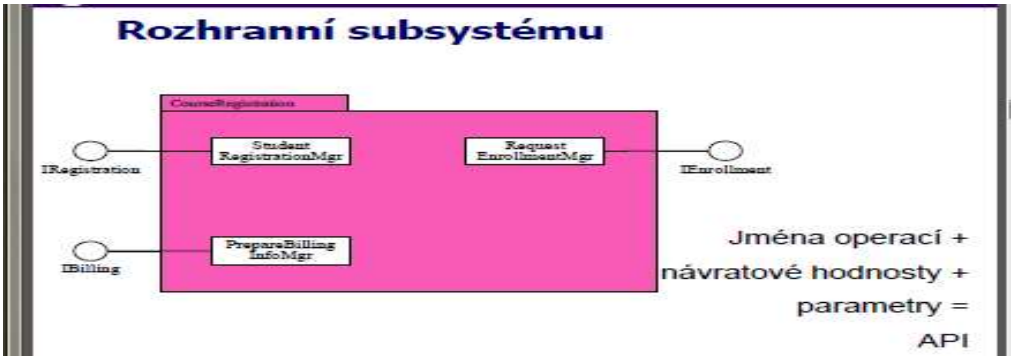
- Objekty z téhož P“ do jednoho subsystému
- Vytvoř speciální subsystém odpovědný za výměnu dat mezi podsystémy
- Minimalizuj počet asociací mezi subsystémy
- Všechny třídy podsystému by měly být funkčně podobné/spřízněné

Návrh subsystémů – vrstvy a členění (partitions)

- **Rekurzivní dělení systému na subsystémy vede k hierarchii subsystémů či vrstev**
- Každá vrstva (subs.) poskytuje vyšší úrovně a využívá služeb nižších vrstev
- **Varianty vrstvené architektury:**
  - **Uzavřené vrstvy**-každá vrstva závisí jen na bezprostředně nižší vrstvě → **nižší provázanost (větší režie)**
  - **Otevřená vrstvená arch.: vrstva může závisle na libovolné nižší vrstvě – větší provázanost (nižší režie)**
- **Členění subsystému na menší celky:**
  - **Dělí služby v 1 vrstvě do dílčích subsystémů**
  - **Výsledky je peer to peer členění (uvnitř vrstvy)**

Vrstvy a jejich členění – příklad





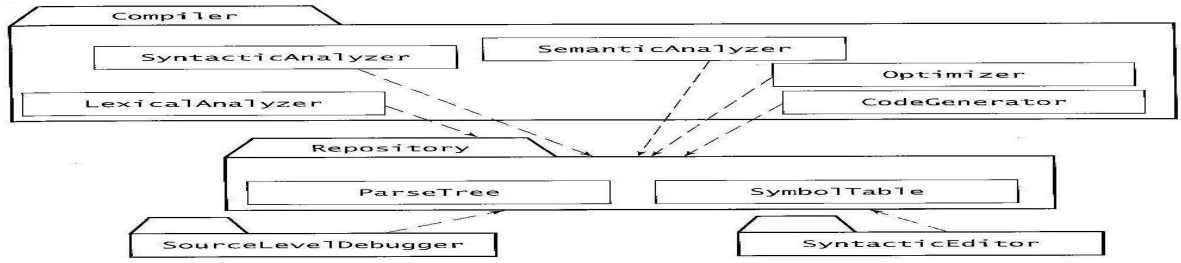
SW architektury

- Rozložení do podsystémů je kritická operace
- Špatně navržené rozdělení se špatně modifikuje
- Proto vzniklo několik SW architektur
- **Architektura zahrnuje:**
  - Dekompozici systému
  - Řízení toku
  - Komunikační protokoly
  - Vymezení hraničních podmínek

Repository

- Základem je sdílená datová struktura – REPOSITORY
- Subsystémy nezávisle interagují s Repository
- Př. Databáze, bankovní systémy
- + sdílení dat (centrální), údržba
- - úzké hrdlo

REPOSITORY – PŘÍKLAD



MODEL/VIEW/CONTROLLER

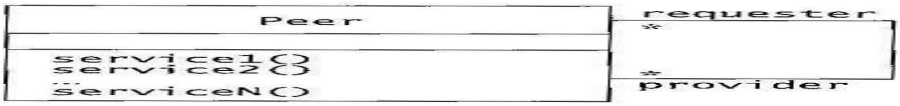
- Model – modeluje doménovou oblast (viz. Analýza a entity třídy)
- View – pohledy na model
- Controller – propojení modelu s uživateli, řízené toku
- + vhodné na interaktivní systémy

KLIENT/SERVER

- Server – poskytuje služby
- Klient – využívá služby
- + vhodné pro distribuované systémy

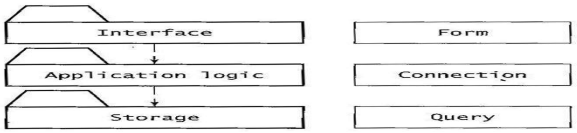
PEER-TO-PEER

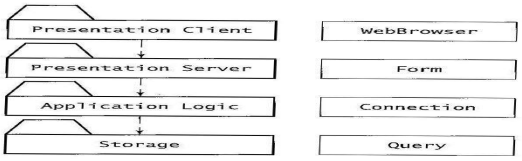
- Zobecnění klient/server
- Peer vystupuje v obou rolích – poskytovatel i klient
- Operace zpětného volání – CALLBACK



TŘÍ-VRSTVÁ,ČTYŘ-VRSTVÁ ARCHITELTURA

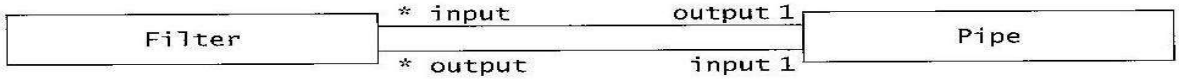
- Častá realizace vícevrstvé architektury
- Odděluje
    - Ukládání dat,
    - Aplikační logiku,
    - Prezentační vrstvy





PIPE & FILTER ARCHITEKTURA

- Každá komponenta definuje formát
    - Vstupy a
    - výstupu
- Komponenty lze propojovat do front (pipe)
- Příklad: příkazy v UNIXU



Zpřesnění návrhu naplňování cílů

Aktivity zpřesňování návrhu

- Výběr hotových komponent a podsystémů
    - Jsou levnější než na míru vytvářený SW
    - Dekompozice je mnohdy potřeba uzpůsobit těmto komponentám
- Mapování podsystémů na hardware
- Návrh infrastruktury zpracování persistentních dat
    - Definice storage-systémů a principů
    - Db, objektová db, soubory
- Specifikace politik přístupových práv
    - Globální access list, table
- Návrh globálního řízení toku
    - Posloupnosti řízení
- Specifikace mezních situací
    - Zapínání/vypínání systémy, restarty
    - Zpracování výjimečných stavů

DEPLOYMENT MODEL (model nasazení)

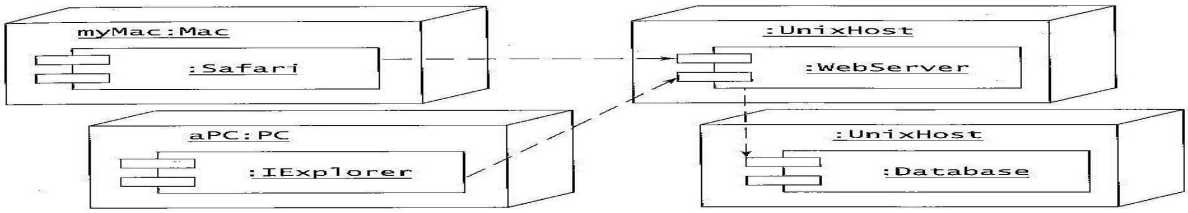
Jméno uzlu	**Distribuce komponent na HW uzlu**

Diagram nasazení zobrazuje:

- **Uzly**: hw zařízení a výpočetní zdroje
- **Vztahy**: komunikace mezi uzly

- Otázky:
  - Kolik a jaké uzly, jaký mají výkon, paměť, ….
  - Propojení, komunikační protokoly,….
  - Vlastností propojení – šířka pásma, dostupnost, kvalita,….
  - Redundance, spolehlivost

DEPLOYMENT – PŘÍKLAD 1



DEPLOYMENT – PŘÍKLAD 2

Další témata zpřesňování návrhu

Management dat – jak jsou zpracována perzistentní data?

Soubory?Relační DBMS? Objektově orientovaná DBMS?

Kontrola přístupu – jak se specifikována a realizována kontrola přístupu?

Globální tabulka práv? seznam práv? schopnost?

Řízení toku – jak je inicializováno a zpracováváno řízení toku?

Procedurální? Událostní model? Vlákna?

Mezní podmínky – jak je řešeno zapínání a vypínání systému a výjimečné stavy?

- Specifikování systémových administrátorských use-casů pro zapínání a vypínání systému
- Specifikování mechanismů ošetřování výjimek pro chybové stavy

Implementační prostředí

Potřeba specifikovat technické a organizační omezení na jejichž základě bude systém vybudován

- Na jakém hardwaru/software systém poběží?
  - Hardware (a jeho omezení)
  - Distribuce SW a HW
  - Systémový SW
- Jaký programovací jazyk bude použit?
  - Objektově orientovaný, ne OO
  - Správa paměti
- Jaký existující SW je potřeba použít?
  - DBMS
  - UIMS
  - Síťový SW
  - Aplikační SW, Framework
- Jací vývojoví pracovníci/organizace se zapojí do vývoje?
  - Distribuovaná pracoviště
  - Týmové kompetence

Návrh tříd – doména řešení

Pro hraniční třídy musíme zvážit
– specifické technologie uživatelského rozhranní
Pro třídy entit musíme zvážit
– specifické technologie správy dat
• návrhové třídy zapouzdřující relační databáze
U kontrolních tříd je třeba zvážit
– otázky distribuce –> potřebujeme samostatnou návrhovou třídu na každém uzlu?
– výkonnostní otázky –> nevyplatí se sloučit některé hraniční/entity třídy?

– transakční otázky –> je potřeba zahrnout technologii správy transakcí?

Návrh tříd

Návrhové třídy jsou takové třídy, jejichž specifikace je natolik kompletní, že mohou být implementovány.

• návrhové třídy získáváme ze dvou zdrojů:

1. problémová doména

• zpřesnění analytických tříd přidáním implementačních detailů

• analytickou třídu může být potřeba rozdělit do více tříd, které ji budou implementovat

2. doména řešení

• třídy z knihoven, znovupoužitelné komponenty, komponentové rámce (DCOM, CORBA, Enterprise JavaBeans, atd.)

• poskytuje technické nástroje pro řežení systému

Návrh tříd — AKTIVITY

• **dokončení specifikace identifikováním/definováním:**

– chybějících atributů, asociací, operací

• ne všechny zprávy se stanou operacemi (např. účastníci, hraniční třídy)

• ne všechny operace se objevují v diagramech aktivit

– typové signatury a viditelnost atributů a operací

– omezení operací –> invarianty; vstupní a výstupní podmínky

– výjimky a výjimečné stavy –> hodnoty, které by neměly operace přijímat

• **výběr znovupoužitelných komponent zahrnuje identifikaci a adaptování:**

– knihoven –> konverzní/"propojovací" třídy a operace (někdy je jich potřeba k propojení a úpravám chování knihovny)

– obecné návrhové mechanismy k vyřežení nefunkčních požadavků

• perzistence; distribuce objektů; bezpečnost; správa transakcí; ožetření chyb, atd.

Návrh tříd — AKTIVITY (pokračování)

• **restrukturování modelu návrhu**

– realizace asociací –> často realizované jako proměnné reprezentující reference (resp. ukazatele) na objekty

• zpřesnění násobností, názvy rolí, asociační třídy, kvalifikované role, orientace asociací; zvážení možností programovacího jazyka

– zvýžení znovupoužitelnosti pomocí dědičnosti a delegování

• **optimalizace modelu návrhu**

– revize přístupových cest s cílem zrychlit přístup k metodám či datům (je možno přidat nové asociace)

– spojení tříd –> třídy s malým počtem atributů a jednoduchým chováním

– cachování náročných výpočtů –> používání odvozených atributů

– odložení drahých výpočtů –> např. zobrazování obrázků, …

Popsáno syntaxí programovacího jazyka.

Vlastnosti dobře navržených tříd návrhu

• **úplnost**

třída by měla dělat, co od ní uživatel očekává – ani méně ani více

• **„primitivnost"**

třída by měla vždy zpřístupňovat nejjednodužší a nejmenší možnou množinu operací

• **vysoká soudržnost**

třída by měla modelovat jeden abstraktní koncept a měla by mít operace, které podporují záměr, pro který byla vytvořena

• **nízká provázanost**

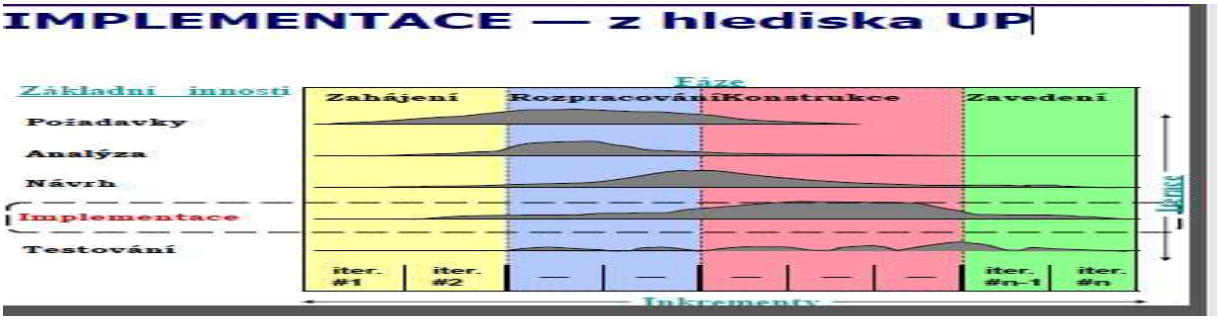
třída by měla být asociována jen a pouze s třídami které jí umožňují realizovat její odpovědnosti

Implementace systému

Účel implementace

Převedení návrhu na funkční (vykonatelný) kód.

- **Workflow implementace** implementuje systém pomocí komponenty jako:
  - Zdrojový kód
  - Skripty
  - Binární soubory
  - Dávky, atp.
- **Je potřeba:**
  - implementovat návrhové třídy a subsystémy z fáze návrhu
  - plánovat integraci systému požadovanou při každé iteraci vývoje
  - integrovat třídy a subsystémy zkompilováním a slinkováním do 1 vykonatelné komponenty
  - distribuovat komponenty na uzly deployment modelu



IMPLEMENTACE – ROLE ŽIVOTNÍHO CYKLU

- **jádrem je fáze konstrukce**
- **vykonávána také během:**
  - **rozpracování: základní skelet architektury**
  - **zavedení: ošetření pozdních chyb a problémů**
- **model implementace je udržován po celou dobu životního cyklu**

VÝSTUPY IMPLEMENTACE

model implementace – popisuje
– způsob implementace modelu pomocí komponent jako soubory se zdr. kódem, dávky, binární soubory, …
– organizaci, strukturu komponent, modularizační mechanismy v implementačním prostředí, programovací jazyk(y)
– závislosti komponent

implementační model je často meziproduktem vzniku zdrojového kódu spíše než samostatnou explicitní aktivitou
expl. modelovací aktivita se děje jestliže:
– kód je generován přímo z modelu ▪ potřeba specifikovat detaily jako komponenty, zdrojové soubory atp.
– jsou-li používány existující komponenty (component based developement
- CBD) ▪ alokace návrhových tříd a rozhranní ke komponentám se stává klíčovým tématem.

VÝSTUPY IMPLEMENTACE – POKRAČOVÁNÍ

- **komponenta** – fyzický balík elementů návrhového modelu – návrh. třídy
  komponenta může implementovat více návrhových elementů

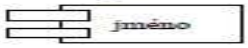
• **implementace subsystémů** – rozdělení implementace do více zvládnutelných celků
• **interface** – totéž jako v modelu návrhu
• **popis architektury** – subsystémy, rozhranní, závislosti a klíčové komponenty modelu implementace
• **plán integrace** – plán konstrukce systému – měl by být inkrementální

LIDÉ

- **architektura** – odpovědná za
  - integritu modelu
  - základní mapování
  - architekturu
- **integrátor** – plánuje buildy, iterace, integraci částí systému
- inženýr komponent – definuje a spravuje zdrojové kódy, soubory, komponenty a integritu subsystémů

Komponenty

Komponenty

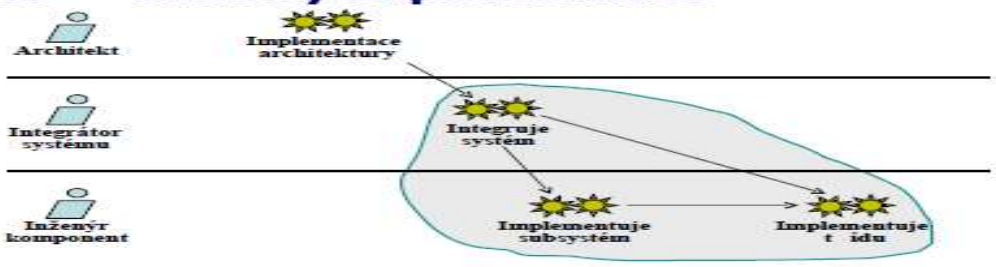


Fyzická, **nahraditelná část** obsahující **balíčky implementace** splňuje **poskytnutou sadu rozhraní.**

- standardní stereotypy komponent:

«executable»	vykonatelná komponenta, program
«file»	soubor se zdrojovým kódem
«library»	statická/dynamická knihovna
«table»	databázová tabulka
«document»	dokument

UP — aktivity implementace



IMPLEMENTACE TŘÍD

Cíl: implementace třídy ve zdrojovém souboru.

Přiřazení návrhových tříd komponentám souborů

– zdrojový kód je v komponentě souboru

– rozsah komponenty souboru které návrhové třídy zahrnout

Rozvržení do souborů závisí na programovacím jazyku a jeho schopnostech.

• Generování zdrojového kódu z návrhu

– cílem je pro každou operaci generovat impl. metodu

– je třeba zvolit vhodný algoritmus a vhodné datové struktury

– generováno na základě: toku událostí, diagramů aktivit, stavových diagramů, diagramu tříd…

Od návrhu k implementace — OO jazyky

MODEL DO PRG. JAZYKA Z NÁVRHU

Analytický model	**Model návrhu**	**C++ zdrojový kód**
Analytická třída	Třída návrhová	C++ třída
Chování třídy	Operace	Členské funkce
Atribut (třídy)	Atribut (třídy)	Statická proměnná
Atribut (instance)	Atribut (instance)	Proměnná instance
Zpráva	Zpráva	Volání metody třídy
Diagram spolupráce	Sekvenční diagram	Sekvence volání
Balíček	Subsystém	Soubory

IMPLEMENTACE SUBSYSTÉMU

- Organizuje výstupy implementace do lépe zvládnutelných celků
  - Může se skládat z komponent, rozhraní a jiných subsystémů
    - **Každý subs. Návrhu je implementován jedním impl. Subs. (vztah 1:1)**
    - **Každý impl. Subs. je obvykle implementován jako komponenta**
- Potřebujeme „balíčkovací mechanismus“ v prg. Jazyce
  - Příklad:
    - **Packegae v Javě**
    - **Project ve Visual Basicu**
    - **Adresář souborů v C++**

INTEGRACE SYSTÉMU

Sestavení (build): implementace části funkcionality systému

- Každý build má kontrolu verzí
  - Umožňuje návrat k předchozí verzi
- Každá iterace živ. Cyklu ve fázi konstrukce odpovídá nejméně 2 sestavení (obvykle však více)

Plán integrace sestavení: sekvence požadovaných builu v rámci iterace

- Specifikuje funkcionality (případy užití/scénaře), které mají být implementovány, subsystémy a komponenty realizující tyto funkcionality.

Obvykle postupuje proces od spodních vrstev a rozšiřuje de do vyšších vrstev (u vrstevnaté architektury).

ZAVÁDĚNÍ (NASAZENÍ) KOMPONENT

SHRNUTÍ

Model implementace obsahuje:
Implementaci subsystémů a jejich závislosti, interface a obsah
Komponenty a jejich závislosti
Mapování vykonatelných komponent na uzly v modelu nasazení
Architektura implementačního modelu

Testování

Úvod: Co to je testování

Proces hledání rozdílů mezi specifikovaným (očekávaným) a pozorovaným (skutečným) chováním systémů.

- Obyčejně realizováno vývojáři, kteří se nepodíleli na implementaci
- Ke kvalitnímu otestování systému je nutné systému dobře rozumět a mít dostatek zkušeností
    - **Nejedná se o práci pro začátečníky**

Cíl: navrhnout testy, které budou systematicky nacházet, chyby Cílem je přimět systém k selhání.

POZNATKY O TESTOVÁNÍ

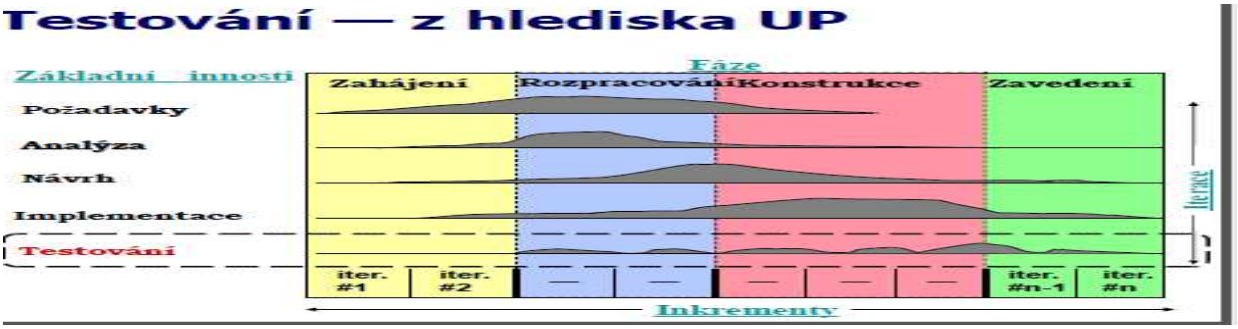
- **Testování neumí prokázat absenci chyb, ale pouze chyby nalézat!**
- **Není možné kompletně otestovat netriviální systém. (většina SW má nutně chyby).**
- **Testování je destruktivní aktivita.**
- **Tzn., kvalitní testování je opačné povahy proti kvatlinímu SW inženýrství**

TESTOVÁNÍ – ZÁKLADNÍ POJMY

- **Spolehlivost –** měří schopnost chování systému dle specifikace
- **Spolehlivost SW –** pravděpodobnost, že SW systém nezpůsobí po stanovenou dobu a ve stanovených podmínkách selhání systému.
- **Selhání –** odchylka systému, po němž chování od očekávaného chování.
- **Chybový stav –** stav systému, po němž následuje selhání systému (manifestace závady).
- Závada (chyba, bug, defekt) – chyba v návrhu nebo v kódu systému, která může způsobit abnormální chování systému.

TESTOVÁNÍ – VERIFIKACE A VALIDACE

- VERIFIKACE je proces ověřování, zda jsme vytvořili produkt správně (tzn. splňující stanované požadavky).
- VALIDACE je proces ověřování, zda jsme vytvořili správný produkt (tzn. naplňující daný účel).
    - Validací se zabývají hlavně akceptační testy
- Testování verifikuje výsledky implementace prostřednictvím testování každého sestavení (build) a finální verze systému
    - Plánování testů je nutné v každé iteraci
    - Návrh a implementace testu zahrnuje
        - **Test. Případy** specifikujícíc co testovat
        - **Test. Procedury** specifikující jak otestovat
        - **Testovací komponenty** sloužící k automatizace testů



VÝSTUPY A PRACOVNÍCI

- **Model testů** – soubor test. Případů, procedur a testovacích komponent popisující způsob testování systému.
- **Testovací případ** – specifikuje způsob realizace jednoho testu
    - Co testovat, vstupy, očekávané výstupy, podmínky realizace
- Testovací procedura – definuje realizace jednoho nebo více testů
- **Plán testů** – popisuje test. Strategie, zdroje a harmonogramy
- **Vyhodnocení testu** – výsledek testovací úsilí
- **Testovací komponenta** – automatizuje jednu nebo více testovacích procedur a jejich částí (někdy se nazývá řadič/ ovládač/driver testu).
- **Defekt** – anomálie systému (např. chyba v programu – bug)

PLÁN TESTŮ

Cíl: **navrhnout sadu testů, která maximalizuje pravděpodobnost odhalení chyb a minimalizuje čas a úsilí.**

Proč? Testování stojí čas a peníze (mnohdy až 40% úsilí bývá věnováno testování).

Strategie testování: specifikuje kritéria a cíle testována

- Jaké testy a jak?
- Míra pokrytí testy (úplnost)
- Síla testů (Kdy skončíme? Požadována míra spolehlivosti).

Odhad potřebných zdrojů: lidských/systémových

Rozvrh testů: kdy který test spustit

NÁVRH TESTŮ

Definuje testovací případy

Testy typu bílá skřínka (White box): „testování v malém“

Verifikace logiky založena na práci se strukturou systému a datovými strukturami → Musí být dostupný zdrojový kód.

(testy vycházejí ze znalosti vnitřního fungování systému/komponenty.)

Testy typu černá skřínka (Black Box): „testováním ve velkém“

Verifikování fungování komponent na základě vstupů a výstupů

→ Zdrojový k=od není nutný!

Regresní (zpětné) testy : selektivní opětovném testování s cílem ověření, že nebyla opravou (úpravami) zavlečena nová chyba.

TESTY TYPU BÍLÁ SKŘÍŇKA – ÚVOD

Cíl: zajistit, že jsme vykonali/prověřili všechny:

- Nezávislé cesty kódem aspoň jednou
- Logická rozhodnutí podle jejich pravdivostních hodnot
- Cykly uvnitř a včetně hraničních podmínek
- Interní datové struktury a jejich validitu

Proč je to důležité?

- Logické chyby a chybné předpoklady jsou nepřímo úměrné pravděpodobnosti průchodu danou cestou v kódu
  - Více chyb v méně používaných částech kódu
- Věříme často, že průchod danou cestou je nepravděpodobný
  - Realita je často neintuitivní
- Překlepy jsou náhodné
  - Netestované cesty pravděpodobně nějaké obsahují

TESTY TYPU BÍLÁ SKŘÍŇKA – PŘEHLED

Můžeme používat následující metody:

1. Test průchodem cest - alespoň jednou projít každou nezávislou cestou
2. Test podmínek - použít každou podmínek s výsledkem true i false
3. Test cyklů - vykonat každý cyklus uvnitř i v jeho krajních podmínek
4. Test datových toků - použít všechny datové struktury a prověřit tak jejich validitu

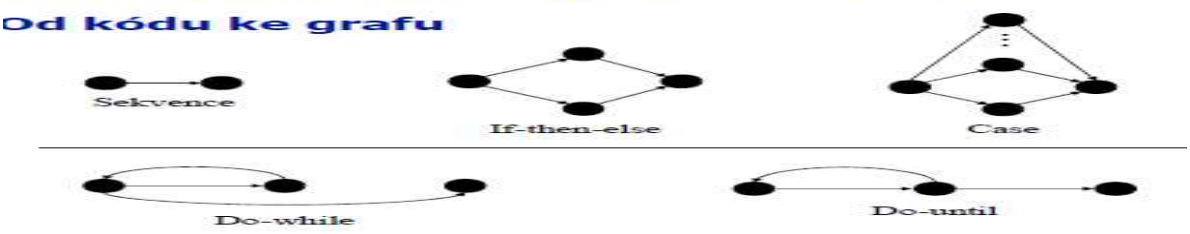
TEST PRŮCHODEM NEZÁVISLÝCH CEST

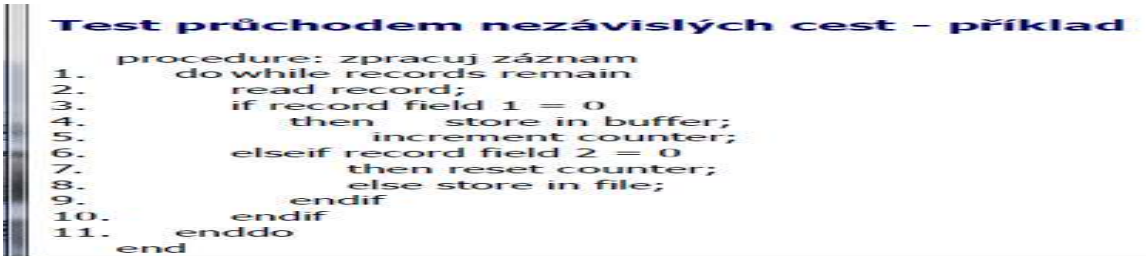
Cíl: alespoň jednou projít každou nezávislou cestou

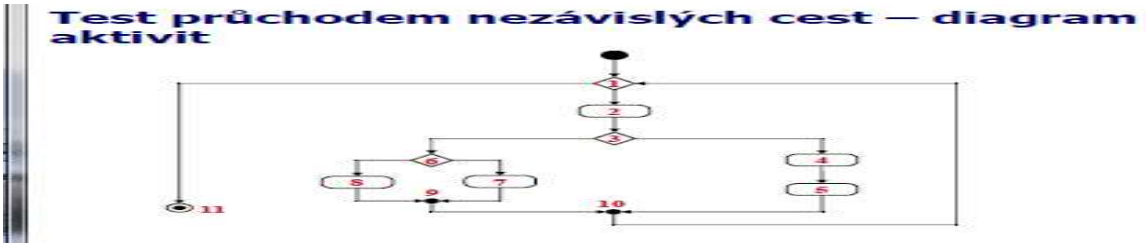
1. Analýza kódu a vytvoření diagramu dat. Toků.
   (pomůckou může být nakreslení diagramu aktivit.)
2. Určení složitosti cyklů v daném kódu
3. Stanovení základní množiny lineárně nezávislých cest.
4. Příprava testovacích případů, které při vykonání vynutí průchodem danými cestami.

Pozor: netestují se všechny možné kombinaci cest. Chceme zajistit pouze průchod každou cestou alespoň jednou.

TEST PRŮCHODEM NEZÁVISLÝCH CEST – 1







TESTY TYPU ČERNÉ SKŘÍNKY

Cílem je zjistit/najít:

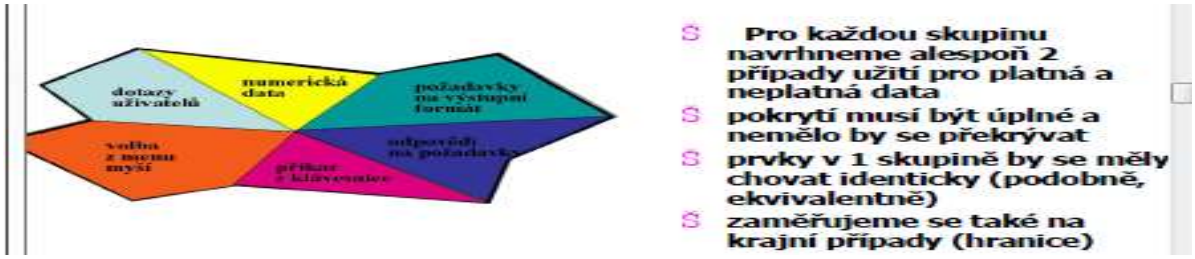
- Nekorektní nebo chybějící funkce
- Chyby nekompability rozhraní
- Chyby dat. Struktur nebo chyby přístup k DB
- Výkonnostní chyby
- Inicializační a koncové chyby

Testovací případy by měly:

- Snížit alespoň o 1 počet jiných potřebných tes. Případů
- Vypovědět něco o (ne)přítomnosti určité třídy chyb
  (např. všechny vstupy v podobě charu jsou zpracovány správně)

Rozdělení na třídy ekvivalence

Rozdělení vstupů a výstupů do skupin tak, abychom pokryli všechna data a třídy chyb.



JINÉ TECHNIKY TYPU ČERNÁ SKŘÍNKA

Odhad chyb

- Volba testovacích hodnot na základě znalostí a zkušeností

Grafy příčin a následků

- Identifikace příčin (vstupní podmínky) a důsledků (akce) subsystému
- Nakreslení grafu
- Převod grafu na rozhodovací tabulku
- Pravidla tabulky jsou převedeny na test. Případy

Srovnávací testování

- V případě vysoké důležitosti se může vyplatit nasazení duplicitních komponent (SW, HW, verze SW)
- Stejná data jsou zpracována různými systémy paralelně a vzájemně kontrolována

POZOR: v případě chybné specifikace duplicity nepomůžou

DOKUMENTACE TEST. PŘÍPADU

Název - jednoznačné jméno k identifikaci

Popis testu – co má test ověřovat

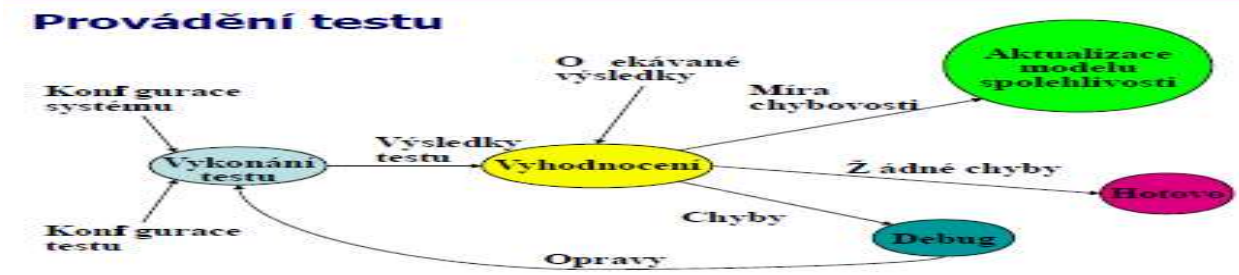
Cílová třída/komponent/subsystém – název testované entity

Testovaná operace – název testované operace

Typ testu – black box; validní/nevalidní vstupy

Testovací hodnoty – vstupy testu

Verifikace - očekávané výsledky



TESTOVACÍ STRATEGIE

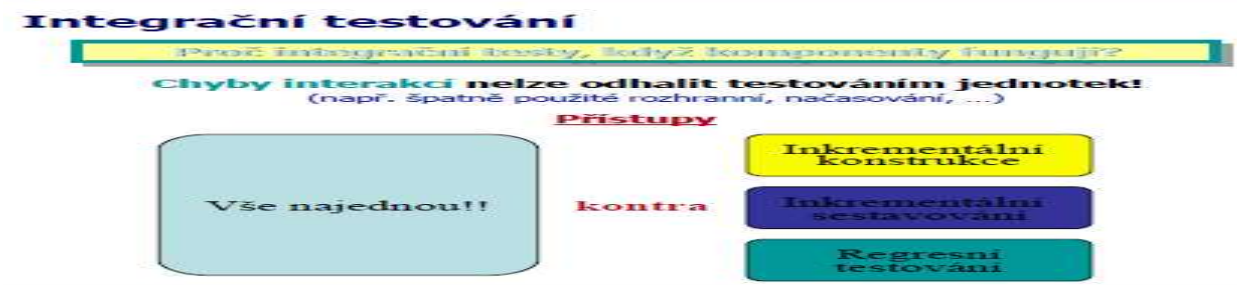
INTEGRAČNÍ TESTOVÁNÍ

Proč, integrační testy, když komponenty fungují?

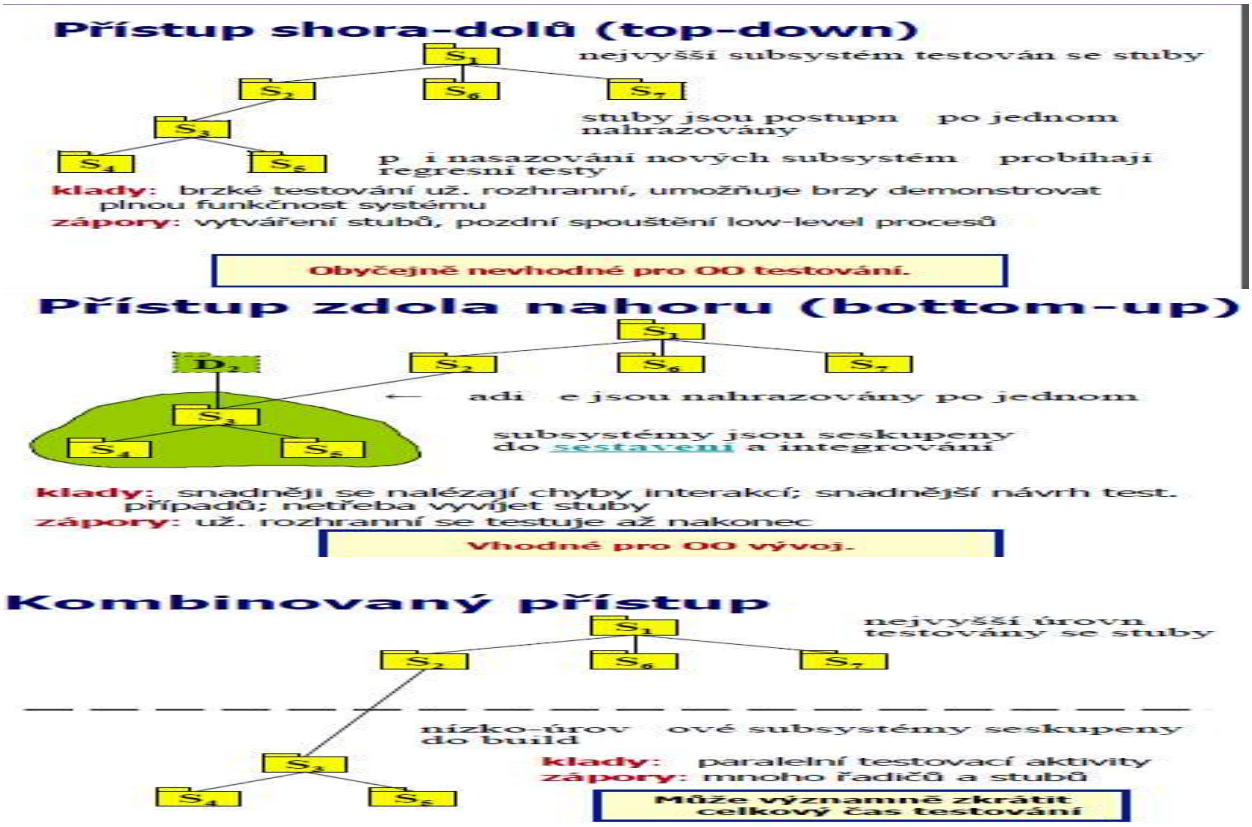
Chyby interakcí nelze odhalit testovacím jednotek?

(např. špatně použité rozhranní, načasování,…)

Přístupy



PŘÍSTUP SHORA-DOLŮ (ROP-DOWN)



SYSTÉMOVÉ TESTOVÁNÍ

Testování systému jako celku.

Specifické typy testů

Funkční – vývojáři verifikují, že všechny uživatelské fce fungují dle specifikace systémových požadavků

Výkonnostní – vývojáři ověřují, zda jsou splněny nefunkční požadavky

Pilotní – vybraná skupiny uživatelů ověřuje běžnou funkcionalitu systému v cílovém prostředí

Akceptační – zákazník ověřuje použitelnost, funkční a nefunkční požadavky dle specifikace systémových požadavků

Instalační – zákazník ověřuje použitelnost, funkční a nefunkční požadavky v reálném použití

TESTOVÁNÍ VÝKONNOSTI

testování „stresu“ ▪ ověření funkčnosti systému v případě mnoha současných požadavků
Kolik systém „ustojí“? Zkolabuje systém nebo skončí korektně?

objemové testy ▪ ověření schopnosti systému zvládat velké objemy dat, náročné algoritmy, nebo velkou fragmentaci disku

bezpečnostní testy ▪ ověření funkčnosti kontrolních a bezpečnostních mechanismů
cena průniku do systému by měla být vyšší než hodnota dat

testy rychlosti ▪ ověření schopnosti systému splňovat časová omezení
obvykle pro real-time a embedded systémy

testy zotavení ▪ ověření schopnosti systému zotavit se po selhání
zotavení je důležité zejména u databázových systémů

AKCEPTAČNÍ TESTOVÁNÍ

Proces zajišťující, že systém odpovídá rozumným uživatelským požadavkům a očekávání

- Může být velice subjektivní – kvalitní specifikace požadavků je důležitá již v ranních fázích projektu.

- Metoda řešení nedostatků by měla být stanovena ještě před fází zavedení
- Akceptační testy (black box) pokrývají následující oblasti:
    - Funkcionalita
    - Výkon
    - Dokumentace
    - Spolehlivost
- Revize konfigurace (audit)
    - Zajišťuje, že byly všechny SW komponenty dodány zákazníkovy (SW konfigurace) správně vyvinuty a zdokumentovány.

VYHODNOCENÍ TESTŮ

- Je třeba
    - Vyhodnotiti výsledky provedených testů
    - Porovnat výsledky s cíli dle testovacího plánu
    - Připravit metriky, které umožní testovacím inženýrům stanovit aktuální kvalitu SW
- Jak poznáme, kdy skončit s testováním?
- Možno zvážit:
    - Úplnost testů: % provedených test. Případů a % testovaného kódu
    - Spolehlivost: porovnání s mírou chybovosti u předchozích projektů (zmenšování v čase)

TESTOVÁNÍ VE STRESU

VŠECHNY TESTY JSOU DŮLEŽITÉ, ALE MNOHDY JE POTŘEBA TESTOVAT JEN OMEZENĚ

- Testování schopností systému (funkcionality) je důležitější než testování komponent
    - Identifikuj funkce, které znemožní uživatelům provádět jejich práci (kritické funkce)
- Testování starých funkcionalit je důležitější než testování nových funkcionalit
    - Uživatelé očekávají, že existující funkcionalita bude nadále fungovat!
- Testování typických situací je důležitější než testování mezních případů
    - Obvykle scénáře použití jsou důležitější než atypické a méně obvyklé scénáře.

SHRNUTÍ

- Je třeba dobře plánovat
    - Musíme vědět, co a proč testujeme
- Použití adekvátních testů
    - White box, black box, regresní testy
- Systematičnost a důkladnost se vyplácí!
    - Jednotkové, integrační, systémové, akceptační testy
- Dopředu se rozhodnout, kolik testování bude prováděno.
    - Definice jasného stop-kritéria