# 1 Základy

*Nesnáším teoretizování o programování, takže možná proto, ale zdá se mi, že v předmětu vyučoval něco moc podrobně, takže tady budou jen informace, které se budu učit na zkoušku (nejde si pamatovat všechno, tak proč bych to sem psal).*

**Softwarové inženýrství** je oblast počítačové vědy, která se zabývá vytvářením softwarových systémů, které musejí tvořit týmy. Velmi se liší od klasického inženýrství.

**Softwarový systém** je systém softwarových komponent (propojená kolekce). Je součástí informačního systému (ten obsahuje i papírování).

**Software** je modelem reality, vývoj SW systémů je tedy hlavně modelování.

**Softwarový projekt** je plánovaná činnost k dodání SW produktu nebo služeb.

# 2 Životní cyklus vývoje software

**Životní cyklus vývoje software** je model procesu tvorby SW systému, definuje postup ve fázích:

1. *Phasing-in* – postupné zavádění produktu (do doby nasazení)
2. *Phasing-out* – postupné vyřazování produktu (od doby nasazení)

**Fáze životního cyklu vývoje software:**

1. *Analýza požadavků*
2. *Návrh systému*
3. *Implementace*
4. *Integrace a nasazení*
5. *Provoz a údržba*

X *Ověřování správnosti a testování* – je v každé části

**Ověření správnosti software:**

1. *Validace* – ověření splnění funkčních požadavků
2. *Verifikace* – ověření dodržení specifikace
3. *Testování* – nalezení chyb
4. *Ladění* – odstranění chyb

**Typy testování:**

1. *Shora dolů* – užití stubs (náhražky podprogramů)
2. *Zdola nahoru* – užití drivers (práce navíc, nadřazené prvky)

1. *Black box* – ze specifikace, test funkčnosti
2. *White box* – z implementace, test kódu

**Údržba kódu**

1. *Opravná* – odstranění chyb
2. *Adaptivní* – přizpůsobení změnám prostředí
3. *Vylepšovací* – rozvoj, nové funkce

**Model Vodopád:**

- Nejstarší, zavedl systematičnost.
- Fáze může začít až po skončení předchozí.
- Plná dokumentace fáze.
- Zpětnou vazbou lze promítnout změny zpět.

**Model Vylepšený vodopád:**

- Fáze těsněji spojeny (překryty) – flexibilnější.
- Zavedeny prototypy jako ukázky (zahodit, nebo dál rozvíjet).

**Nevýhody vodopádů:** nelze paralelizovat, malá zpětná vazba, plánování prostředků je na počátku (nepřesné).

**Model Iterativního cyklu s přírůstky:**

- Rozkládá proces na iterace, každá předkládá prototyp.
- Iterace přidávají přírůstek (nové funkce).
- Krátké iterace vedou k lepší kontrole a plánování.
- Většinou několik vodopádů v iteracích.

**Spirálový model:**

- Iterace dělena na 4 kvadranty.
- Plánování, odhad rizik, inženýrství, hodnocení.
- V části inženýrství je vodopád, jinde je možnost zastavit práci, pokud je špatná/nevýhodná.

**MDA** (Model Driven Architecture) využívá specifikaci, kterou pomocí CASE nástrojů převede na implementaci. Ne vše lze takto provést, zatím v počátku, používá často UML.

**Agilní programování** je moderní přístup k rychlému vývoji, vývoj ve dvojicích, společný kód, upřednostňuje se implementace před dokumentací apod.

# 3 Modelování struktury v UML 2.0

**Diagram tříd** vizualizuje třídy, rozhraní a vztahy mezi nimi. Asociační třída je pro vztah N:N.

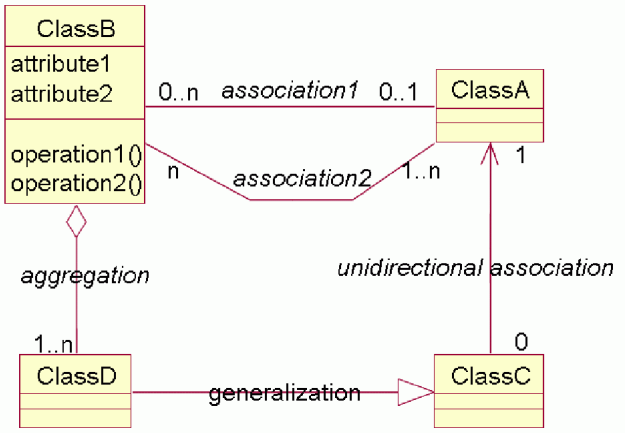



**Diagram objektů** ukazuje objekty a jejich vztahy v jistém časovém okamžiku (snapshot). V obrázku je i odpovídající diagram tříd.

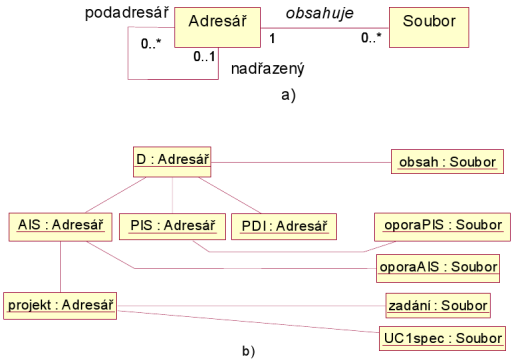



**Diagram balíčků** modeluje balíčky – logické seskupení prvků modelu, dobré ke strukturování.

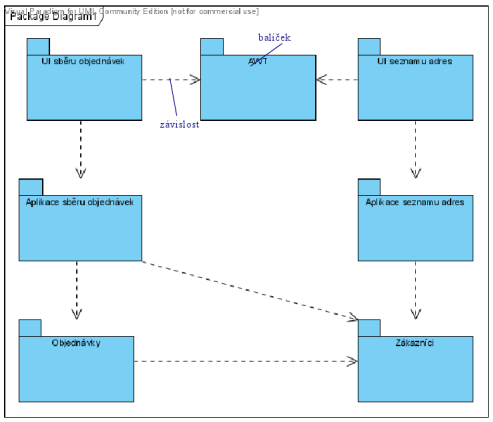



**Diagram komponent** ukazuje závislosti mezi SW komponentami a jejich implementací.

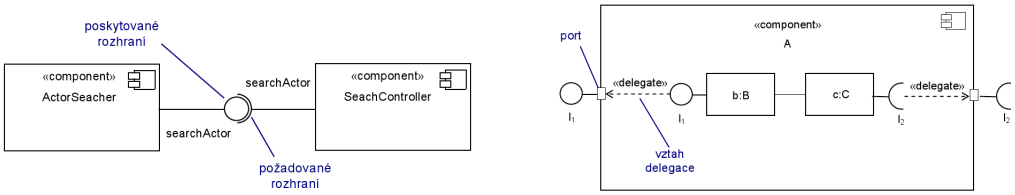


**Diagram složené struktury** ukazuje propojené prvky systému za běhu, které spolupracují k dosažení cíle. Zachycuje statickou strukturu beze zpráv.

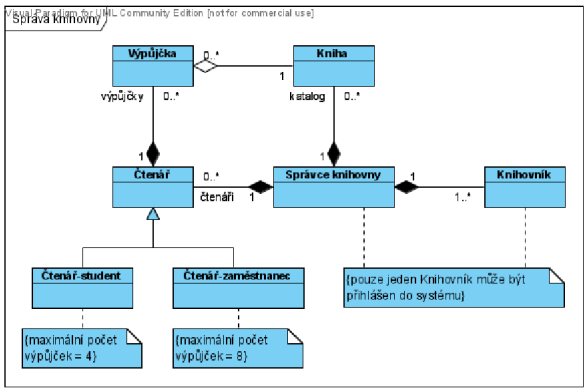
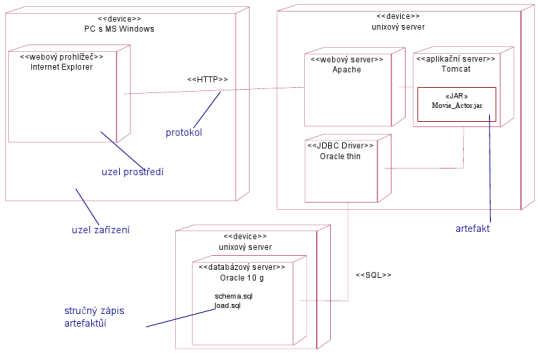
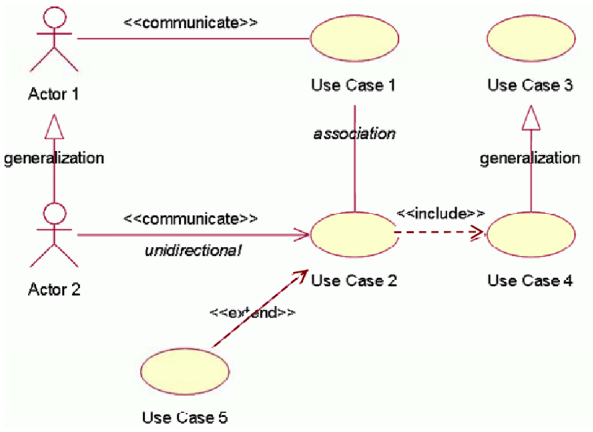



**Diagram nasazení** ukazuje nasazení komponent na fyzickou architekturu a prostředí (HW a SW).




# 4 Modelování chování v UML 2.0

**Diagram případů použití** specifikuje funkčnost systému. `<<include>>` říká, že funkce je zahrnuta v jiné, `<<extend>>` říká, že může být použita v jiné. Součástí diagramu je také textová specifikace případů.




**Diagramy interakce** jsou obecný název pro další techniky návrhu chování, modeluje interakce tříd.

**Diagram sekvence** obsahuje účastníky, jejich čáru života (čas) a dobu, kdy jsou aktivní. Ukazuje předávání řízení zprávami (synchronní i asynchronní), alternativní tok pomocí bloku `alt`.

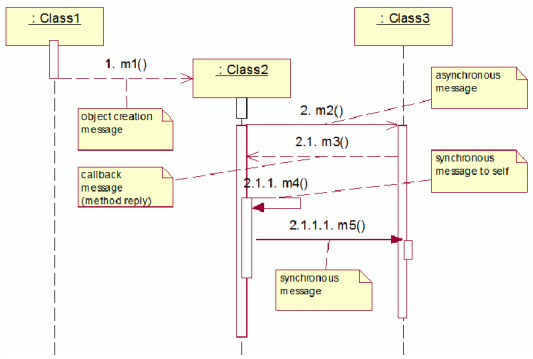
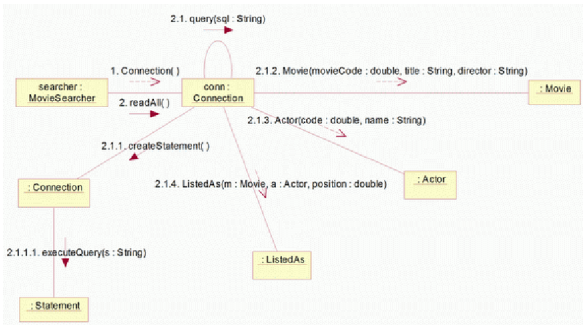



**Diagram komunikace** je podobný sekvenčnímu, ale bez druhé dimenze, ukazuje statické vazby, které se využijí při komunikaci.




**Přehledový diagram interakce** speciální forma diagramu aktivity, interakce a jejich výskyt.

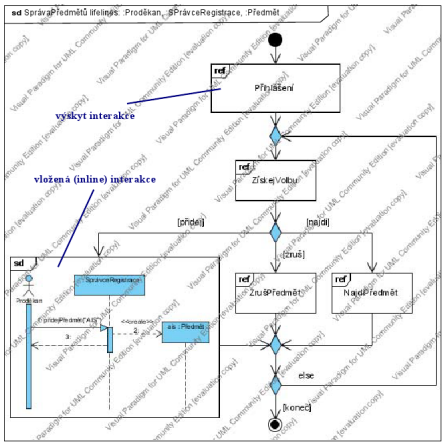



**Diagram časování** je v UMLv2, diagram změn v čase.

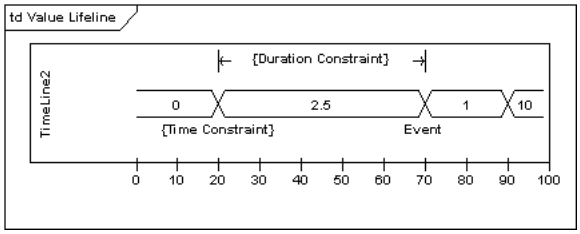



**Diagram stavového automatu** obsahuje stavy a přechody, které jsou spojeny s akcí/podmínkou.

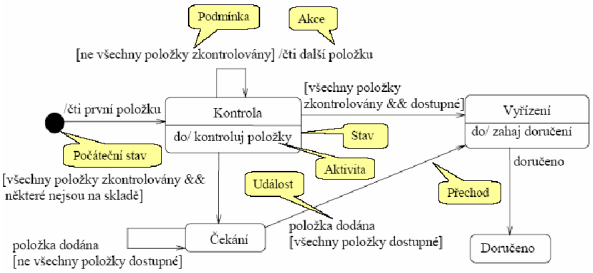
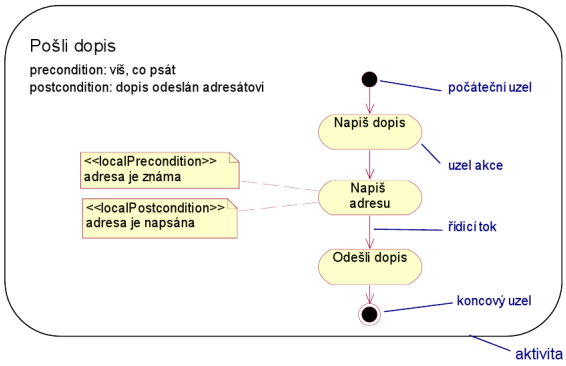



**Diagram aktivity** vychází z Petriho sítí, modeluje aktivitu, prostředky i pro paralelní aktivity. Může obsahovat oddíly (aktivita v Praze, Brně, ...), řídící uzly (podmínka, cyklus, fork, join).




# 5 Úvod do plánování a sledování projektu, řízení projektu

**Projekt** je časově ohraničená aktivita k dosažení cíle.

**Řízení projektu** zahrnuje strategii a plánování, základem plánování je trojimperativ *kdy, co, za kolik*, tedy čas, kvalita a náklady.

**WBS** (Work Breakdown Structure) je členění práce na části a milníky.

**Milník** je kontrolní bod, významná událost o nulovém časovém trvání.

**Fáze** je ohraničená časová část projektu.

**Etapa** je skupina fází.

**Zdroje** zahrnují pracovníky, vybavení a materiál.

1. *Pracovní* – znovupoužitelné, mají dostupnost a vytížení (lidé, stroje)
2. *Materiálové* – spotřebovávají se

**Plánování:**

- Rozdělení činností ideálně po 14ti dnech.
- Cíl je pohyblivý, podmínky se neustále mění.
- Nejrizikovější jsou lidské zdroje.
- *Kritická cesta* je sekvence úkolů, které při pozdržení pozdrží celý projekt.
- *Kritický úkol* je úkol na kritické cestě, má vyšší prioritu.

- Mezi úkoly jsou závislosti a hierarchie.
- *Ganntův diagram* je hlavním prostředkem vizualizace plánování.

**Plánování řízené úsilím** přiděluje úkoly zdrojům a to tak, aby maximalizoval využití zdroje (jeho úsilí). Je-li využit málo, dostane další úkol, je-li využit moc, přidělí se další zdroj.

**Plánování rozpočtu projektu** se snaží odhadnout cenu projektu předem. Některé metody jsou přesnější, jiné levnější, čím blíže konci, tím také přesnější odhad.

- *Expertní odhad* – založen na zkušenostech, více expertů zlepšuje, tak kvalitní jako expert, pro větší projekty nepoužitelné, jinak poměrně kvalitní
- *Analogie* – podle jiného projektu, ale je třeba mít podrobná data (problém)
- *Řízeno plánem* – sumarizace cen za jednotlivé úkoly
- *Algoritmus* – specializované modely pro výpočty cen

**FPA** (Function Point Analysis) odhaduje rozpočet z funkčních bodů (vyjadřují práci za úseky v člověkohodinách) plus opravné faktory.

**COCOMO** (Constructive Cost Model) odhaduje rozpočet z počtu řádků kódu, přesnost se hodně liší na době použití ($0.25 \rightarrow 4$ krát), program Costar.

**Ostatní algoritmy:**

- *Pricing to win* – cena je taková, kolik je zákazník ochoten dát, zbytek ve službách.
- *Karnerova metoda* – počítá se z pohledu zákazníka pomocí UC diagramu.

**Sledování průběhu projektu** využívá prostředky plánování, ale pravidelně kontroluje jejich dosažení nebo rozdíl skutečnosti a plánu.

**Řízení lidských zdrojů** se stará o organizaci lidí, nábor nových pracovníků, rozvoj týmu a motivaci.

**Motivace:**

- *Motiv* – vnitřní impuls
- *Podnět* – vnější impuls
- *Motivace* – snaha ukojit potřebu, ovlivněna motivy a podněty
- *Stimulace* – snaha upravit sílu motivu
- *Maslowova motivační teorie* – hierarchie potřeb (od jídla po obdiv), nižší se musejí uspokojit aby byla touha po vyšších.
- *Dvoufaktorová teorie* – satisfaktory (vnější) a motivátory (vnitřní), ke spokojenosti stačí satisfaktory, ale k motivaci i motivátory.
- *Vroomova expektanční teorie* – člověk musí věřit, že práce přinese výsledek, který bude odměněn a o tu odměnu musí stát.
- *Teorie tří faktorů* – existenční, vztahové, růstové.

**Komunikace** je nejdůležitější v managerské oblasti, několik úrovní (informační, motivační, kontrolní, emotivní), komunikace meziúrovňová (šéf a zaměstananec), potřebná při řešení konfliktu.

**Řízení rizik** se snaží nalézt, analyzovat a reagovat na rizika. Snaha maximalizovat pozitivní výsledek situace a minimalizovat dopad negativního. Tvoří se protiriziková opatření a operativně se řídí.

**Protiriziková opatření:**

- *Předcházení* – eliminace příčin
- *Zmírňování* – snížení dopadu nebo pravděpodobnosti
- *Aktivní přijetí* – plán ošetření následků
- *Pasivní přijetí* – smíření se s následky

**Řízení kvality** musí být v každé fázi projektu a odděleně (vlastní plán a rozpočet). Snaha standardizovat.

**CMM** (Capability Maturity Model) má pět úrovní vyspělosti tvorby software, poslední úroveň odpovídá ISO 9001:2000.

**Diagram příčin a následků (Ishikawův)** studuje příčiny (boční čáry) problému (přímá hlavní čára), jde dále delit.

# 6 Úvod do metodiky Unified process

**Unified Process** je generický proces vývoje SW, adaptuje se pro dané účely a organizaci (pro firemní standardy, nástroje, šablony, databáze apod.).

**Základní axiomy:**

1. Je řízený požadavky a riziky.
2. Staví na robustní architektuře systému.
3. Je iterativní a inkrementální. Každá iterace zahrnuje všechny fáze běžného vývoje (plánování, analýza, konstrukce, integrace, testování).

**Baseline** je výsledek jedné iterace.

**Fáze vývoje** obsahuje jednu nebo několik iterací (a mezivýsledků – baseline). Má určeny cíle a zaměřuje se nebo zdůrazňuje některé činnosti.

1. *Zahájení* – stanovení proveditelnosti, vytvoření bussiness případu, zachycení požadavků, identifikace rizik
2. *Rozpracování* – spustitelná verze architektury, zpřesnění rizik a požadavků, definice kvality, plán konstrukce a prostředků
3. *Konstrukce* – doladění požadavků, implementace
4. *Zavedení* – opravy chyb, příprava pracoviště k nasazení, přizpůsobení SW a pracoviště, manuály, konzultace

**Rational Unified Process** je komerční implementací obecného Unified Process. Rational Software byl koupen IBM. Využívá CASE.

# 7 Podstata a metody určení požadavků

**Určení požadavků** je hlavně manažerská činnost pro získání specifikace požadavků na produkt od zákazníka. Je potřeba hluboké pochopení požadavků, jinak nastanou nákladné změny v budoucnu.

**Funkční požadavky** – co má systém dělat: vstupy, výstupy, nabízené funkce, ...

**Nefunkční požadavky** – jak to má systém dělat: použitelnost, znovupoužitelnost, spolehlivost, efektivita, bezpečnost, podporovatelnost, ...

**Získávání požadavků** je proces komunikace mezi analytikem a zákazníkem, výsledkem je obchodní model použití (upravený use-case) a diagram tříd.

**Tradiční metody** jsou jednodušší, levnější, efektivní pro menší projekty

- *Rozhovor se zákazníkem* → požadavky na použití
- *Rozhovor s odborníkem v oblasti* → znalost problematiky
- *Dotazníky* – více času, méně ostychu, ale klient se nemůže zeptat
- *Pozorování* – sledování procesu, aktivní zapojení nebo s doprovodem
- *Studium dokumentů organizace*

**Moderní metody** jsou složitější, dražší, vhodné pro velké a rizikové projekty

- *Prototypování* – nejpoužívanější, zákazník dostane celý systém, ale bez plné funkčnosti
- *Brainstorming* – konference s moderátorem, více nápadů, odstraňuje rozpory
- *Společný vývoj aplikací (JAD)* – moderátor, písař, zákazníci a vývojáři
- *Rychlý vývoj aplikací (RAD)* – upřednostňuje rychlost před kvalitou

**Vyjednávání a validace požadavků** jsou potřeba kvůli nejasnosti nebo nereálnosti požadavků (mohou se překrývat, být v rozporu). Také je potřeba eliminovat nepotřebné požadavky nebo požadavky mimo oblast. Využívá se matice požadavků, sestavuje se žebříček priority požadavků.

**Správa požadavků** se stará o identifikaci, klasifikaci, organizaci a dokumentaci požadavků. Každý požadavek má jednoznačné ID, vhodné při použití CASE. Požadavky tvoří hierarchii a je potřeba je verzovat.

**Obchodní model požadavků (BOM)** je modelování procesů ve firmě, pro kterou se tvoří produkt. Modeluje pro zákazníka, zjednodušuje, zavádí obchodní slovníček.

**Donénový model požadavků (DOM)** modeluje oblast (doménu) bussiness procesu, kterého se projekt týká přímo. Používá konkrétnější diagramy než BOM a rozšiřuje slovníček.

**Obchodní slovníček** ustanovuje pojmy pro obě strany, důležitý pro jednoznačnost na obou stranách. Může být postaven na jiném slovníčku, ale je nutná revize. Při budování se pojmy přidávají v každé fázi projektu.

**Model rozsahu systému** je DFD řádu 0, tedy pouze systém a jeho okolí (není součástí UML).

**Obchodní model použití** je use-case diagram, ale na vysoké úrovni abstrakce.

**Obchodní diagram tříd** je opět více abstraktní model diagramu tříd, třídy zde jsou spíše vyšší entity, jejichž atributy nejsou plně důležité.

**Dokument požadavků** je hlavní a zásadní výstup procesu určení požadavků. Obsahuje soupis všech funkčních a nefunkčních požadavků, specifikuje cíle projektu, rozsah apod. Většinou se tvoří na základě standardizované šablony (IEEE aj.).

# 8 Závislosti vrstev, balíčků a tříd

**SW architektura** je způsob uspořádání SW komponent tak, aby splňovalo různé požadavky (funkční a nefunkční).

**Vrstvy** SW architektury napomáhají lepší pochopitelnosti systému, přehlednosti a udržovatelnosti. Dovolují přímou komunikaci pouze v rámci vrstvy, vynucují viditelné závislosti.

**Závislost** entity A na entitě B znamená, že změna v B může vynutit změny v A. Ve správné architektuře by neměly být závislosti mezi nesousedícími vrstvami a cyklické závislosti.

**Balíček** (dle UML) je seskupení elementů modelu pod jedním jménem. Může obsahovat jiné balíčky, členové pod něj spadají (odebráním balíčku se odeberou i členové). Může importovat ostatní entity.

**Odstranění cyklické zavíslosti mezi balíčky** A a B se provádí přidáním speciálního balíčku A2, který obsahuje prvky A takové, že je na nich B závislé. Na novém balíčku A2 je pak závislé A i B. Odstranil se tak jeden směr závislosti, druhý zůstává nezměněn.

**Závislost vrstev** by měla být směrem od vyšší k nižší. Nižší vrstvy by měly být *stabilní* (neměnit se příliš v čase).

**Závislost tříd** je hlubšího rázu, třída většinou závisí na třídě v jiném balíčku nebo vrstvě a tím zesložiťuje závislosti balíčků a vrstev.

**Závislost metod** vzniká voláním v kódu.

**Odstranění cyklických závislostí tříd** pracuje na principu vytvoření nového rozhraní, na které se závislost předá (tzv. Presenter). Je také možné vytvořit rozhraní pro snížení počtu závislostí.

# 9 Vrstvy a architektonické rámce (MVC, PCMEF)

**MVC** (Model-View-Controller) je model architektury, která rozděluje systém na tři části (M V a C).

- *Model* reprezentuje data
- *View* reprezentuje způsob zobrazení dat
- *Controller* reprezentuje komunikaci v systému

**PCMEF** (Presentation, Control, Mediator, Entity, Foundation) je model architektury, tvoří hierarchickou strukturu a vývojář všechny komponenty musí přiřadit některé z nich.

- *Presentation* – UI, interakce s uživatelem
- *Control* – zpracování žádostí, vnitřní komunikace
- *Mediator* – prostředník mezi DB a pamětí (programem)
- *Entity* – správa objektů v paměti
- *Foundation* – základní komunikace s DB (dotazy apod.)

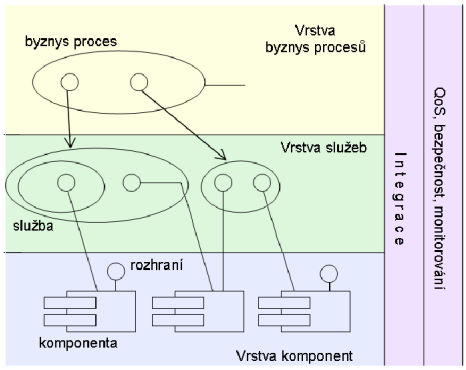
**Principy PCMEF:**

1. Princip závislosti směrem dolů
2. Princip sdělování směrem nahoru
3. Princip komunikace sousedů
4. Princip seznamovacího balíku
5. Princip explicitních asociací
6. Princip eliminace cyklů
7. PRincip pojmenování tříd (P,C,M,E,F)

# 10 Servisně-orientovaná architektura

**SoA** (Service-oriented Architecture) je jistým stylem vytváření informačních systémů, kteý reflektuje bussiness proces. Model je založen na komunikaci mezi poskytovatelem služeb a spotřebitelem služeb, bývá rozšířeno o registr služeb (podle něj si spotřebitel služby vyhledá).

**Struktura SoA** je mnohovrstevnatá:




**Spolupráce mezi službami** – služby nekomunikují pouze se spotřebitelem, ale i mezi sebou:

- *Kooperace* – služba využívá prostředky jiné služby
- *Agregace* – služba tvořená pouze jinými službami (spojení), sjednocení výsledků
- *Choreografie* – spolupráce služeb napříč organizacemi

**Vlastnosti SoA:** volné propojení, nezávislost služeb, abstrakce, znovupoužitelnost, bezestavovost, nezávislost na platformě

**SoA vs. Klient-Server** – nesouvisejí, SoA je plně distribuovaná, spotřebitel je také služba (druhé strany), K-S jsou jednostranné a propojené dvě entity.

**SOAD – Servisně orientovaná analýza a návrh** je moderním postupem návrhu systému, kombinuje OO, rámce pro podnikovou architekturu a bussiness process modelling.

**Webové služby** jsou reálnou implementací SoA, založeny na XML, SOAP, UDDI a WSDL.

**WSDL (Web Services Description Language)** – W3C standard pro popis webových služeb

**SOAP (Simple Object Access Protocol)** – W3C standard pro komunikaci mezi službami a mezi poskytovatelem a spotřebitelem

**UDDI (Universal Description, Discovery and Integration)** – adresářová služba, bez standardu, už nemá veřejně přístupné rejstříky

# 11 Úvod do rámců pro webové aplikace

**Rámce pro webové aplikace** jsou prostředky (knihovny a nástroje) pro vývoj webových aplikací a služeb.

**Microsoft .NET Framework SDK** – integrace do VS, *Compact framework* pro mobilní aplikace, podpora Office, služby implementovány pomocí ASP.NET

**Sun Java EE** využívá HTTP servlety (na straně serveru převezmou požadavky v HTTP) a data předává objektům implementujícím službu. Podpora XMP, SOAP, RPC.

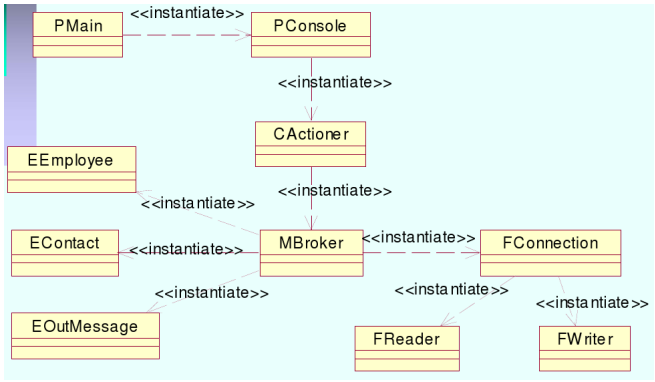
**JAX-WS** (Java API for XML Web Services) poskytuje prostředky pro realizaci XML služeb (přenos XML zpráv SOAP pomocí HTTP). Služby poskytovatele jsou zapsány jako Java třídy, WSDL popis se generuje z rozhraní.

# 12 Návrh tříd a interakcí, GRASP

**Návrh tříd** je proces, který zajišťuje, že třídy poskytují chování žádané v use-case diagramu a odpovídají návrhovým rámcům. Zahrnuje také návrh rozhraní a je neoddělitelný od návrhu interakcí.

**Návrh interakcí** rozšiřuje návrh tříd o detaily jako jsou operace a metody, přidává sekvenční a komunikační diagramy.

**Instanciace tříd** se zabývá otázkou, kdo volá konstruktor. Některé jsou vytvořeny při spuštění, nekteré mají známého původce při překladu, ale někdy je tato vazba dynamická. Existují pak instanciační diagramy.




**Interakce** jsou realizovány jako sekvence zpráv (synchronních i asynchronních) mezi životními čarami (lifelines).

**GRASP** (General Responsibility Assignment Software Patterns) je princip přiřazení zodpovědnosti třídám a objektům podle vzorů:

- *Information Expert* – centrální prvek, který má důležité informace a proto i zodpovědnost
- *Creator* – třída, která zakládá objekty jiné třídy; důvod: obsahuje/agreguje/často používá danou třídu

- *Controller* – prvek, který řídí systémové události (není v UI)
- *Low Coupling* – přidělovat zodpovědnost tak, aby bylo co nejméně závislostí
- *High Cohesion* – udržovat kohezi (soustředění závislostí na jednom místě)
- *Polymorphism* – je-li chování závislé na typu, zodpovědnost dát polymorfním operacím
- *Pure Fabrication* – speciální třída, která se použije pro dodržení výše uvedených pravidel (pokud nejde aplikovat na existující třídy)
- *Indirection* – třída, která dělá prostředníka a zamezuje přímému propojení
- *Protected Variations* – zabránení šíření změn, např. pomocí rozhraní

# 13 Návrhové vzory

**Návrhové vzory** jsou osvědčená řešení typických OO problémů. Jsou popsány a katalogizovány pro snadné použití.

**Adapter** (také Wrapper) obaluje rozhraní třídy tak, aby ji bylo možné použít s jiným rozhraním.

**Abstraktní továrna** je centrální objekt pro vytváření nových objektů.

**Singleton** – třída může mít jedinou instanci. Pak je možné mít pevný bod přístupu k objektu.

**Pozorovatel** (také Publish-Subscribe) objekty se mohou registrovat jako pozorovatelé změny stavu daného objektu.

**Fasáda** poskytuje jediné rozhraní pro celý subsystém (několik tříd s rozhraními).

**Řetěz zodpovědnosti** zahrnuje příkazové objekty a zpracovávací objekty, zapojené za sebou (analogie zřetězených procesorů).

# 14 Refaktorizace

**Refaktorizace** je změna zdrojových kódů tak, že se nezmění chování systému, ale pouze jeho vnitřní struktura.

**Přínosy:** Vyčištění kódu od dočasných nebo starých konstrukcí. Zlepšení designu nebo postupu, optimalizace. Přejmenování pro vhodnější pochopení.

**Hlavní oblasti**

- Duplicitní kód
- Dlouhé metody
- Velké třídy
- Příliš mnoho parametrů
- Odstranění závislostí pro změny (shotgun surgery)
- Data, která jsou používána na mnoha místech a měla by být objektem
- Feature envy – čtení dat z mnoha jiných objektů k vlastnímu výpočtu

**Metody refaktoringu:**

- *Extract class* rozděluje velkou třídu na několik menších, původní třída je pak volá.
- *Extract interface* tvoří více rozhraní pro jednu třídu namísto jednoho velkého.
- *Subsume method* eliminuje metodu jejím začleněním do jiné, vhodné pro duplicitní výskyty kódu.

# 15 Zajištění bezpečnosti dat

**Nepovinná autorizace** (DAC) spočívá v přidělení privilegií uživatelům, ti tak získají přístup k daným prostředkům.

**Autorizace pomocí rolí** (RBAC) je DAC, kde se privilegia přidělují rolím a ty pak uživatelům, jde pouze o vrstvu, která usnadňuje správu a zvyšuje přehlednost.

**Povinná autorizace** (MAC) přiděluje uživatelům přístupovou úroveň a objektům bezpečnostní třídy. Systém pak obsahuje pravidla přístupu pro dvojice třída-úroveň.

**Deklarativní nepovinná autorizace** přiděluje privilegia pro jistou akci na daném objektu (SELECT na tabulce X). Lze použít i role, důležitá je možnost revokace privilegií.

**Programová nepovinná autorizace** je pokročilé nastavení privilegií, lze je omezit na pohled, synonyma, skrýt logickou strukturu DB (vytvoří se pohled a pouze ten se zpřístupní; zpřístupní se jen synonymuma tím se skryje struktura). Je možné použít databázové procedury, na ně stačí privilegium EXECUTE, není třeba přístup k datům.