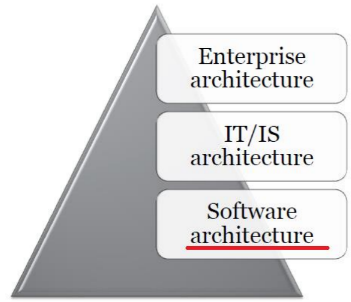
NSS přehled

Návrhové vzory a logická architektura systému. Modely chování, vzory chování, use-case. Rozhraní systému. (NSS)

# Verzování

* Čísla verzí se skládá většinou z 3 čísel : X.Y.Z
* První číslo je hlavní (X), druhému se říká vedlejší (Y) a třetímu se číslo revize (Z)
* Pokud se zvýší nějaké číslo tak čísla nižší se resetují (př. : 1.0.1 se zvýší na 1.1.0)
* Číslo revize se zvyšuje po pár commitech (opravy menších chyb)
* Pokud už je velké množství revizí, tj. zásahů, při kterých se pouze opravuje kód, přijde doba, kdy je kód potřeba celý přepsat, zpřehlednit a něco málo upravit → zde se zvyšuje vedlejší číslo
* Hlavní číslo se zvyšuje při kompletně přepracované programu, který již má odlišnou podobu od předchozí verze

# Softwarová architektura (SA)

* SA je struktura, která se skládá z komponent a vztahů mezi nimi:
  + komponenty, konektory, konfigurace
* Není zcela libovolná, ale je omezená pravidly, účelem a prostředím.
* **Příklady**: SoC (Separation of Concerns), Call and return, Objektové paradigma, Klient-Server, Návrhové vzory, Komponenty, Roura (pipes and filters), ESB

# Informační architektura (IA)

* Model, který zahrnuje všechny aspekty informačního systému dané organizace, např. software, hardware, organizaci, procesy
* **Informační systém** je systém složený z osob, datových záznamů a aktivit, které zpracovávají data a informace v organizaci, což zahrnuje jak manuální, tak automatizované procesy
* Samotné UML není na této abstrakci již dostačující
* Informační systém nejsou jen počítače, ale i celá organizace získávání, zpracování a distribuce údajů (užitečných informací) k uživatelům.

# Enterprise architektura (EA)

* **MIT definice**: Jedná se o organizační logiku podnikových procesů a IT infrastruktury odrážející integraci a standardizaci požadavků firemního operačního modelu.
* Pojem EA představuje architekturu systému na nejvyšší úrovni. Její součástí (nervovou soustavou) je informační architektura. Vyjádření architektury softwarových komponent, ze kterých se informační systém podniku skládá je pak softwarová architektura.
* SA –> IA –> EA -> BA

Business architektura (BA)

* Reprezentuje model, který popisuje tok informací a práce mezi autoritami (velmi obecné)
* Tato architektura nám musí odpovědět na základní otázky:
  + Co děláme, Kdo to dělá, Kterou informaci, Kde je to uděláno?
* Využívá různé principy/frameworky :
  + Object Management Group (OMG)
  + Unified Modeling Language (UML)
  + Model Driven Architecture (MDA)
  + Business Motivation Model (BMM)
  + Business Process Modeling Notation (BPMN)

# Další dělení architektury

**Statická**

* Zachycuje pevnou danou strukturu softwaru
* Nemění se za běhu, nemůžeme spoléhat na učení za běhu
* machine learning apod.

**Dynamická**

* Oproti statické architektuře navíc podporuje vznik a zánik komponent a vazeb za běh struktura systému se dynamicky mění
* Např. architektura distribuované aplikace, která si dynamicky alokuje v síti dostupné zdroje během výpočtu, tj. architektura se sice mění, ale je dopředu daná

**Mobilní**

* Rozšiřuje dynamickou architekturu o mobilní prvky, kdy se komponenty a vazby přesouvají za běhu systému podle stavu výpočtu
* Např. aplikace s multiagentní architekturou, kde je změna struktury součástí výpočtu), tj. architektura se mění a není dopředu předvídatelná

# ADL = Architecture description language

* Musí umět vyjadřovat komponenty, konektory a konfigurace systému (důležité!)

**Základní elementy architektury**

* **Komponenty** –části dekomponovaného systému s daným rozhraním
* **Konektory** –komunikační kanály pro propojení komponent s daným rozhraním
* **Konfigurace** –konkrétní způsob vzájemného propojení komponent pomocí konektorů
* Má podobně vlastnosti jako UML
* Jedná se o jazyk pro popis softwarové, technické nebo systémové architektury
* **Systémová architektura** je konceptuální (pojmový) popis struktury a chování systém
* **Technická architektura** je struktura HW komponent systému a SW komponent na nich běžících.
* **Softwarová architektura** je struktura systému složeného ze softwarových komponent, jejich vlastností a vztahů mezi nimi.

**Formální:** př.: Wright

* Využívají matematické, algebraické nástroje, grafy apod.
* Jsou přesné, ale horší na používání a pochopení

**Neformální/obecné**: Př. ACME, AADL, Archimate

* Čitelnější, některé jsou omezenější například jen na embedded systémy

# GRASP = General Responsibility Assignment Software Principle

GRASP, consist of guidelines for assigning responsibility to classes and objects in object-oriented design

1. Information Expert
   * Kdo za co zodpovídá, princip přiřazení odpovědnosti
2. Creator
   * Kdo vytváří nové instance / Kdo by měl vytvářet
3. Low Coupling
   * Malá závislost mezi komponenty – malý dopad při změně, lepší přepoužití
4. High Cohesion
   * Všechno, co k sobě patří u sebe – zvladatelná komplexita při rozšíření
5. Controller
   * UI Event controller – řízení systémových událostí
6. Polymorphism
   * Overload, Override
7. Pure fabrication
   * Abstrakce – oddělení od reálnýho světa -> zvýšení cohesion a snížení coupling
8. Indirection
   * Neptat se na přímo (možno na stejný vrstvě) – získání informace pomocí delegace přiřazeným objektům
9. Dont talk to strangers
   * Jasně dáno, s kým můžu komunikovat

# Cache

**Příklady** technologií: Hazelcast, Infinispan, JBoss Tree Cache

**Lze využít k:**

* Optimalizaci výkonu systému
* Snížení náročností jednotlivých operací
* Snížení náročností na jednotlivé vrstvy
* Mitigaci rizika krátkodobé nedostupnosti systému

**Typy cache:**

* **Nodové**
* **Clusterové**: distribuované, replikované

**Data v cache**

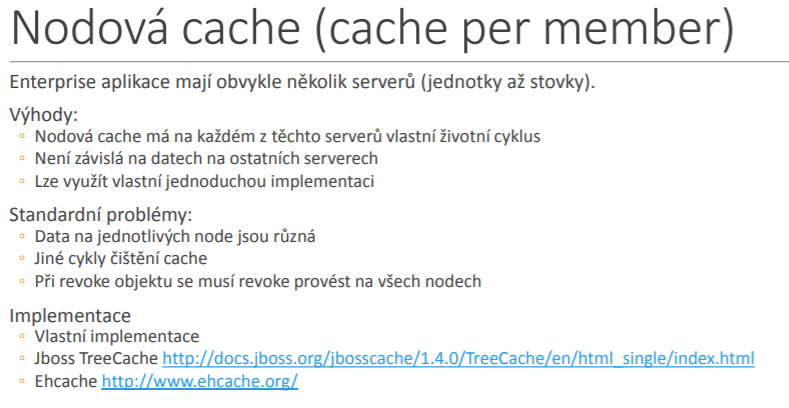
* *Co můžeme mít v cache*: Jakékoliv objekty, Výsledky SQL dotazů (MyBatis Cache per select)
* *Co raději ne:* Velké obrázky, Videa
* *Obsahově***:**
  + Statická data
  + Data, která se minimálně mění
  + Data, která je výpočetně těžké získat
  + Data, ke kterým přistupujeme opakovaně

**Typy úložišť pro cache**

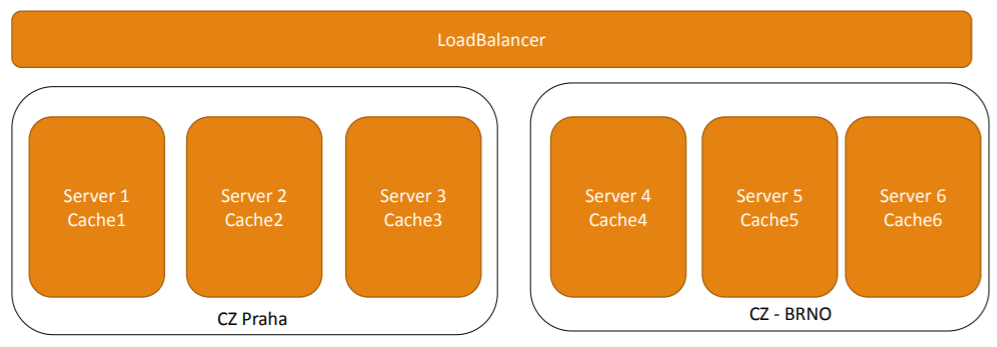
* In-memory
  + Nejčastější použití data se načtou do paměti
* FileSystem
  + Zejména pro velké soubory, statické obrázky a pod
  + Data jsou na File Systému mimo aplikační server.
  + Zátěž se přesouvá na úložiště.
  + Ideálně v kombinaci s CDN
* Databáze
  + Data mohou být uložená v DB
  + V případě, kdy potřebujeme replikovat data z jiného systému
  + Popřípadě jsou zde již transformovaná, přepočítaná, agregovaná data

**Základní operace**

* Načtení dat do cache (Write)
  + Lze vložit všechna data, lze vkládat data postupně
* Přístup k datům v cache (Read)
  + Na základě identifikátoru, hashe objektu, ….
* Obnova dat v cache (Aktualizace dat)
  + Obvykle aplikační záležitost
  + Z principu použití cache je těžké říci kdy aktualizovat a jak
* Revokace objektů (Čištění cache)
  + Automatická – po vypršení TTL, po určité době, kdy nebyl objekt přečten, po přeplnění vyhrazené kapacity
  + Časová – automatizovaný JOB vyčistí celou cache

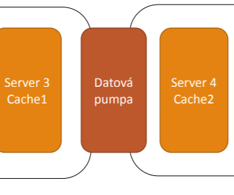
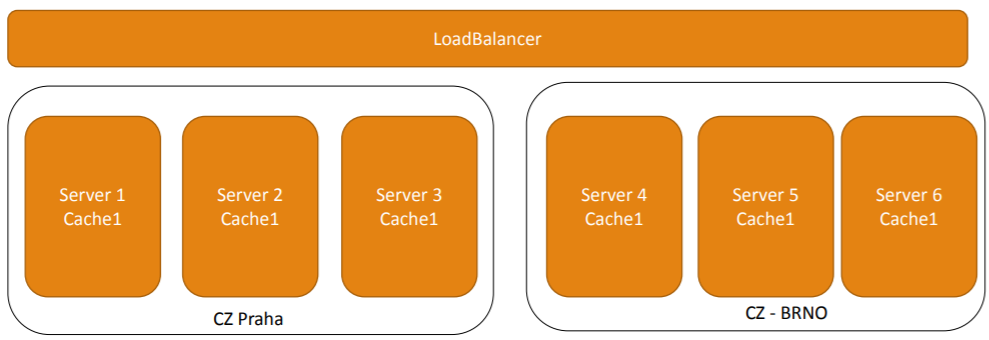


Implementace: Vlastní, Jboss TreeCache, Ehcache

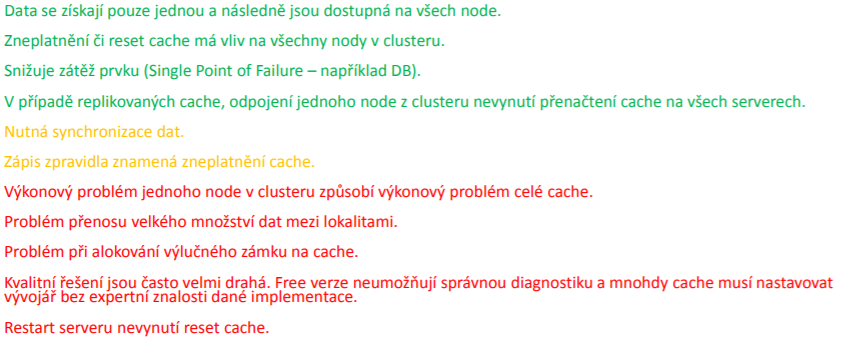




Implementace: inifinispan, hazelcast, Jboss TreeCache, Redis



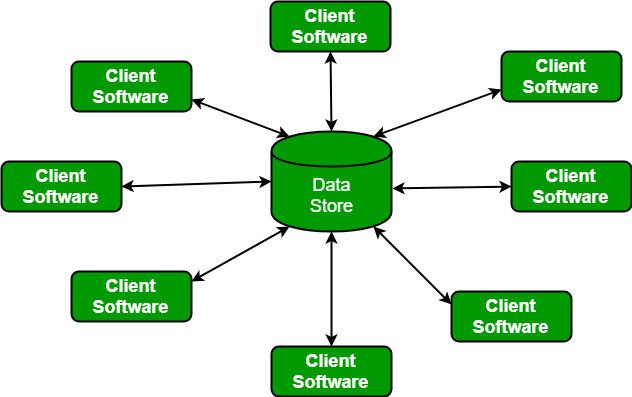
**Výhody a nevýhody clustrové cache**



# SW Architektura

## **Data centric**

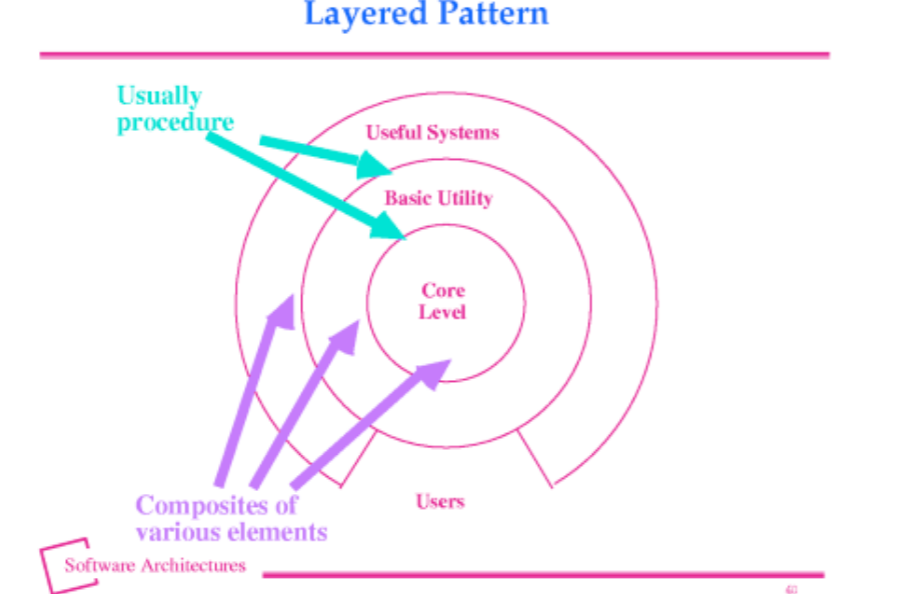
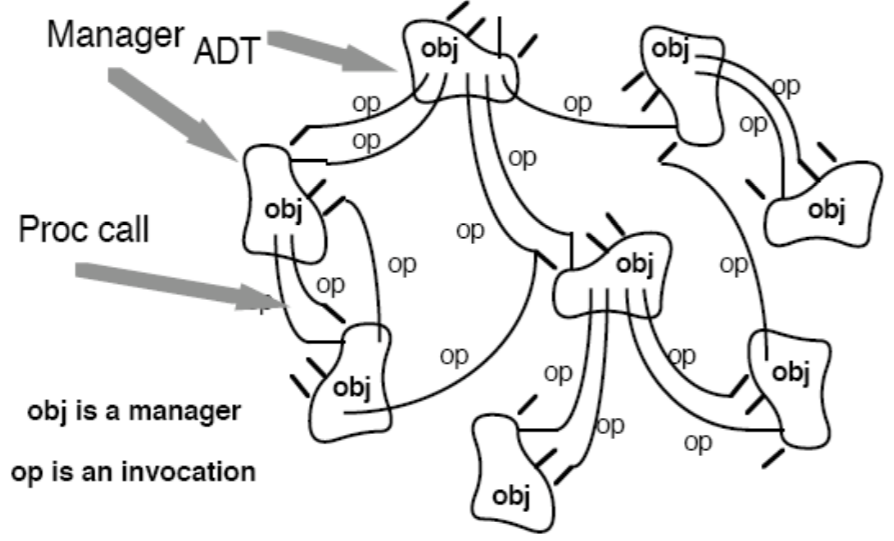
Databases, Voice recognition, Compilers

* Všechny aplikace v organizaci používají **společný data model**
*  **Vlastnosti:** Data integration, Distribution, Control, Coordination, Scalability, Low coupling, Centralization, Reuse, Modifiable,

## **Call and return**

OOD, Procedural, RPC, AOP, Layers, RMI

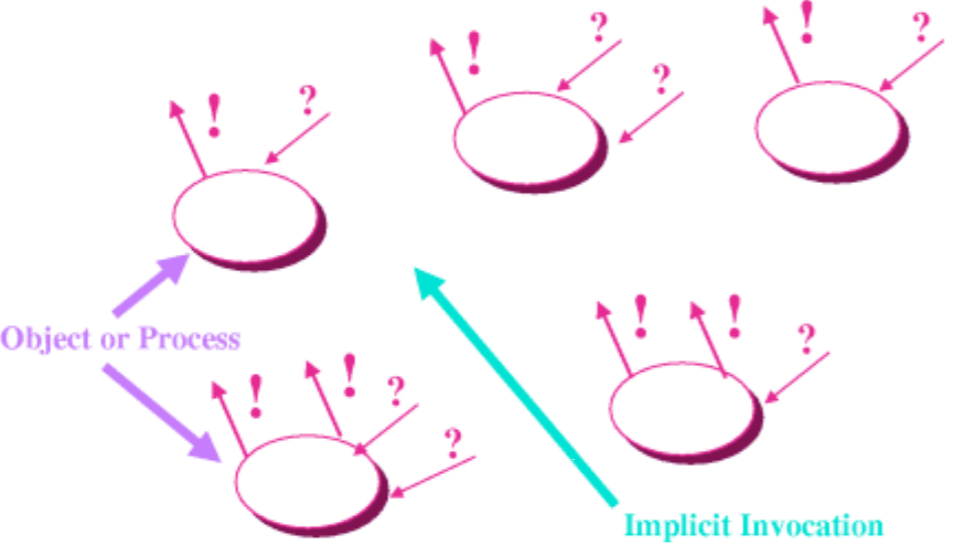
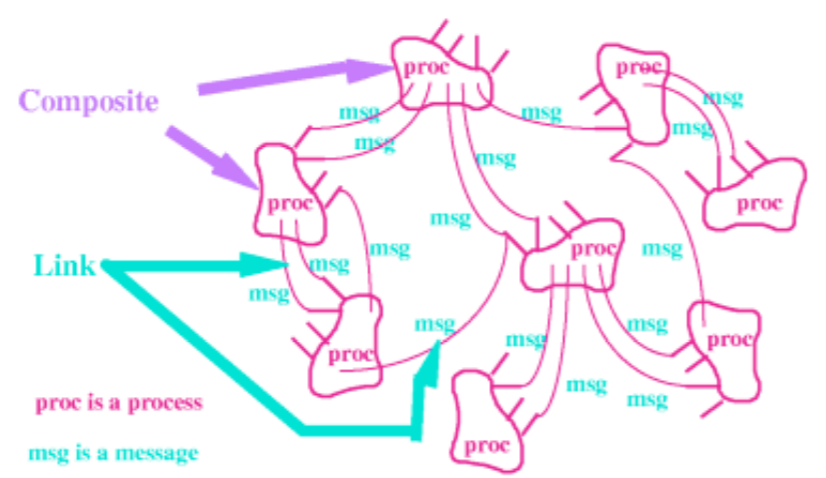
* V podstatě se zavolá main funkce která invokuje jiný subfunkce napříč stroji atd..
* **Vlastnosti:** Modifiable, Reusable, Inf. hiding, Structural decomposition, Separation of concerns

## **Implicit** **invocation**: **Events**

* Event handling za pomocí callbacků – IoC
* **Vlastnosti:** Modifiable, Low coupling, Hard to comprehend,

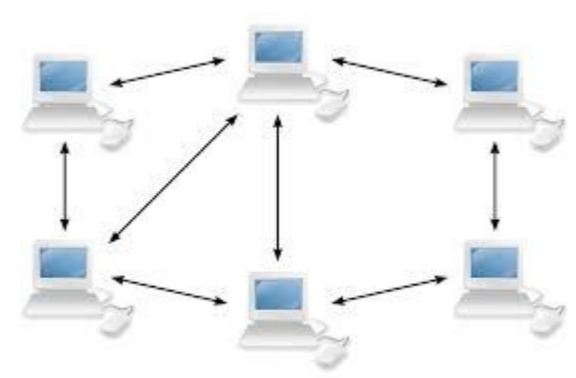
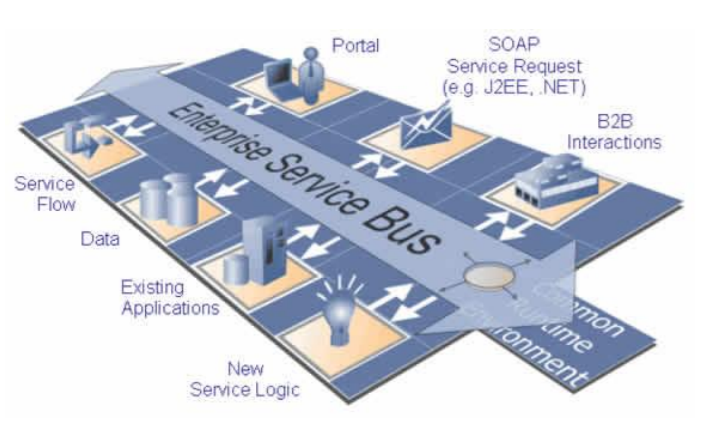




## **Independent components**

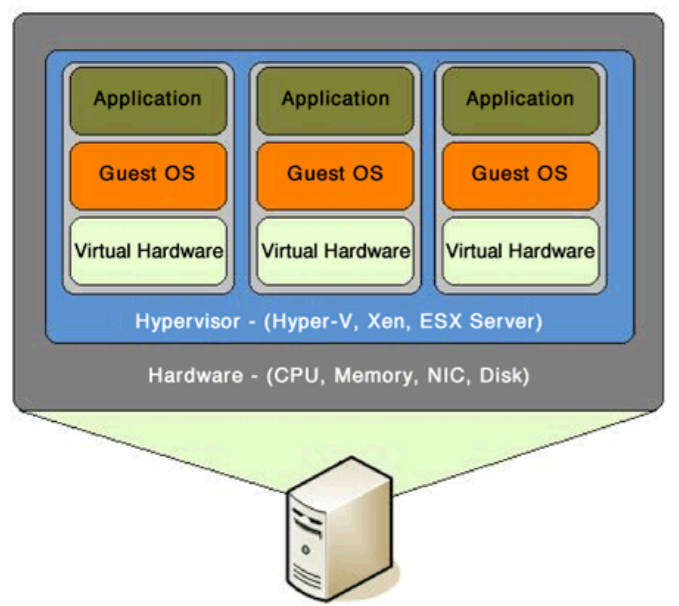
ESB(SOA), peer to peer

* **Vlastnosti:** Integration, Scalability, Reuse, Low coupling, Distribution, Reliability



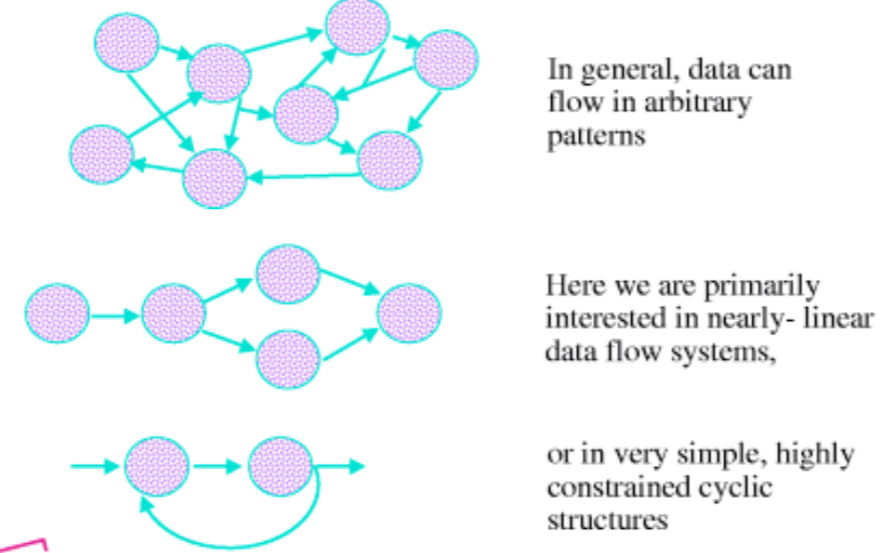
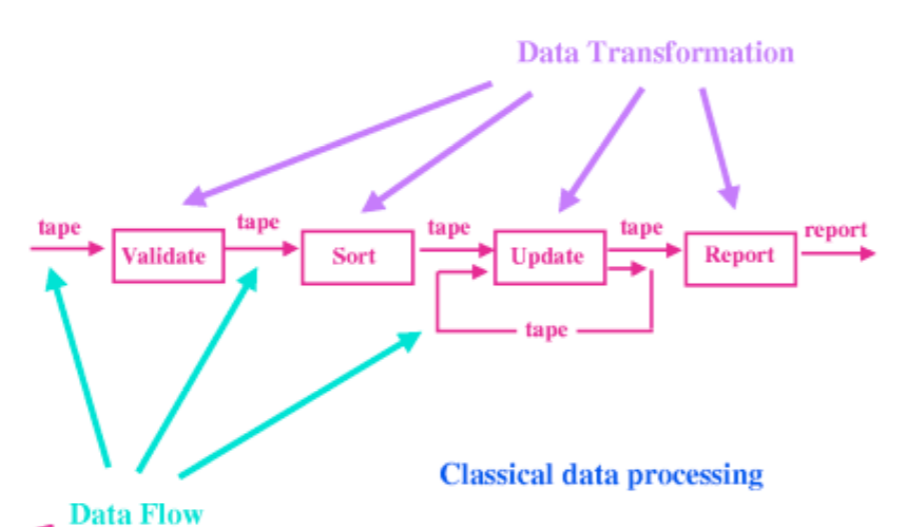
## **Virtual Machines**

* **Vlastnosti:** Simulation, Emulation, Portability!, Flexibility, Lowered Performance, Extended features



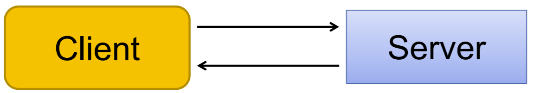
## **Pipe and Filter**: data flow

* Populární architektura pro Compiler, UNIX Shell (např. cat, grap a sort kombinace) a např. Java Stream
* **Vlastnosti:** Modifiable, Reuse, Easy design, Simplicity, Low Coupling, Slow, No filter cooperation, Lot of parsing



## **CLIENT-SERVER ARCHITECTURE**

**Properties**: Centralization, Easy with security, …

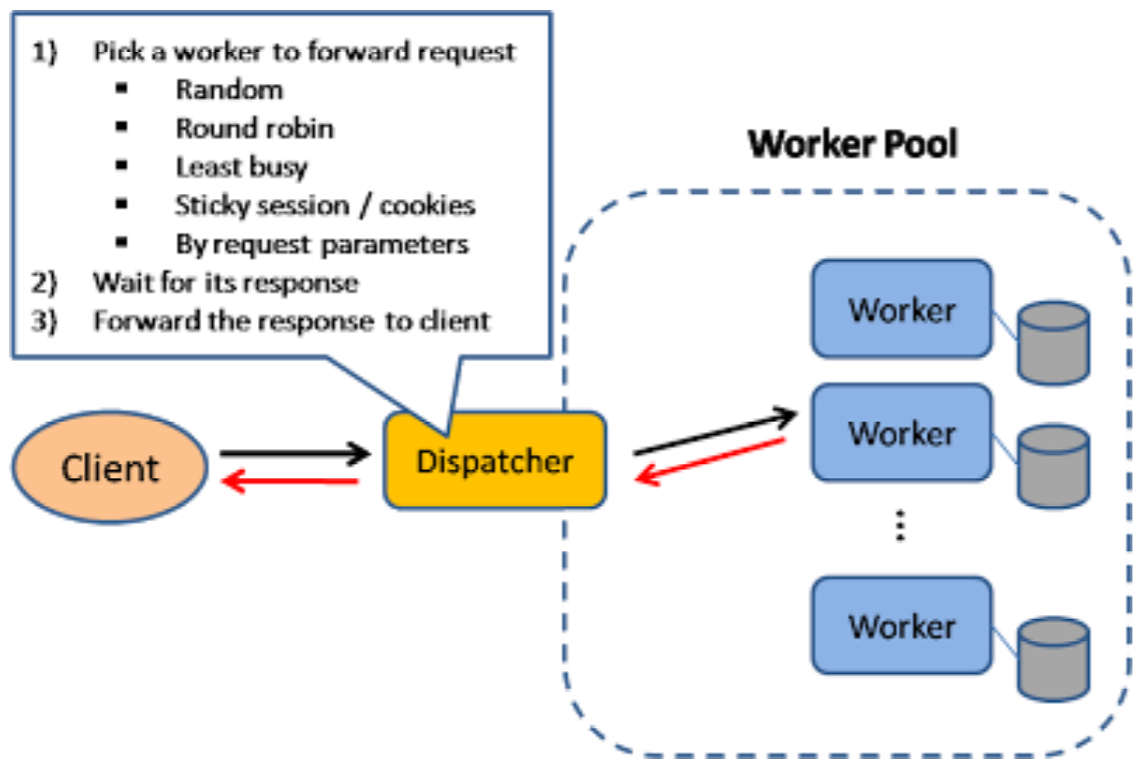
* Easy to scale
  + Until we reach the limit
  + Server is the bottleneck
* Performance influenced by the network conditions
  + And virtual distance between client and server
* Server has given throughput
  + Given by HW, our Design, Efficiency, Caching, etc.

**Jak zlepšit výkon:**

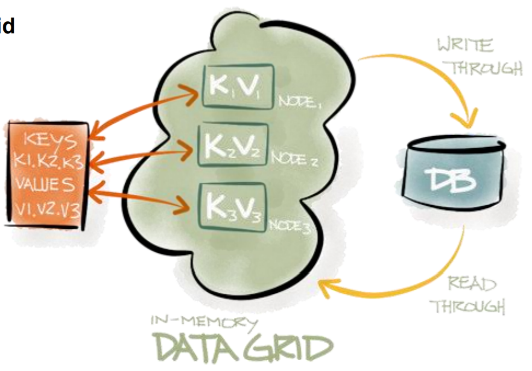
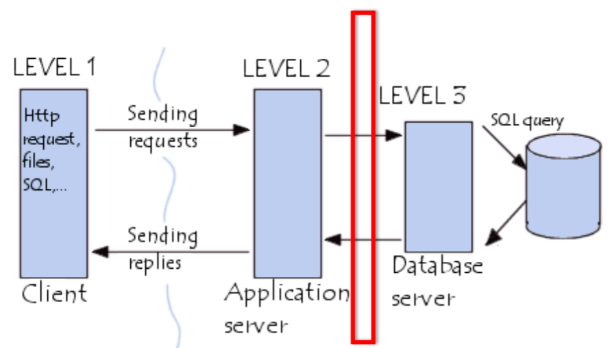
* Caching
* Performance analysis – profiling
* Native/Custom SQL queries for reports
* Better Hardware, more CPU/Mem

**Dispatcher**

* Filtr (loadbalancer), který bude filtrovat jednotlivé requesty a bude je rozdělovat mezi připojené servery, které můžou býti statické nebo dynamické



### Database bottleneck



## Content delivery systém - CDN

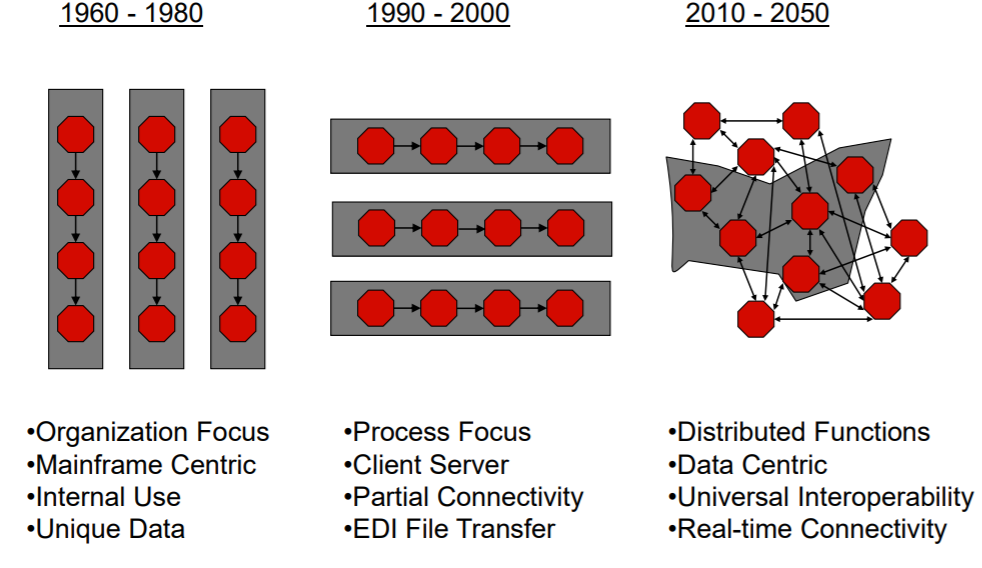
Síť pro doručování obsahu Content Delivery Network neboli CDN je síť vzájemně propojených počítačů skrze internet, která umožňuje dostupnost obsahu nebo dat (obvykle velký multimediální obsah) uživatelům.

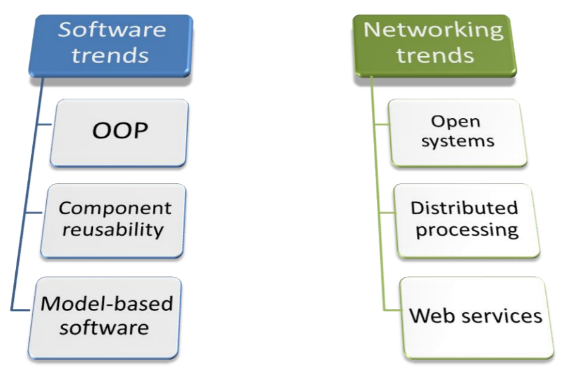
Tato síť se skládá z:

* Zdrojového serveru, který poskytuje obsah pro CDN.
* Prvků rozmístěných v různých částech internetové infrastruktury, kde je obsah replikován.
* směrovacího systému, který zajišťuje, aby byl obsah uživateli doručen z geograficky nejbližšího uzlu sítě.

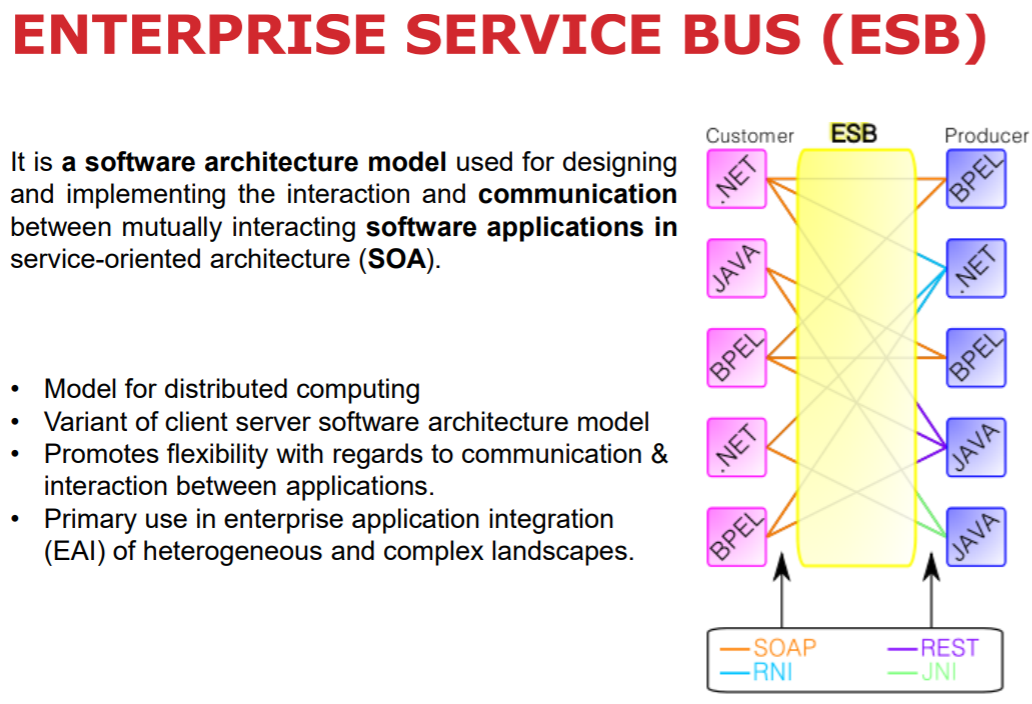
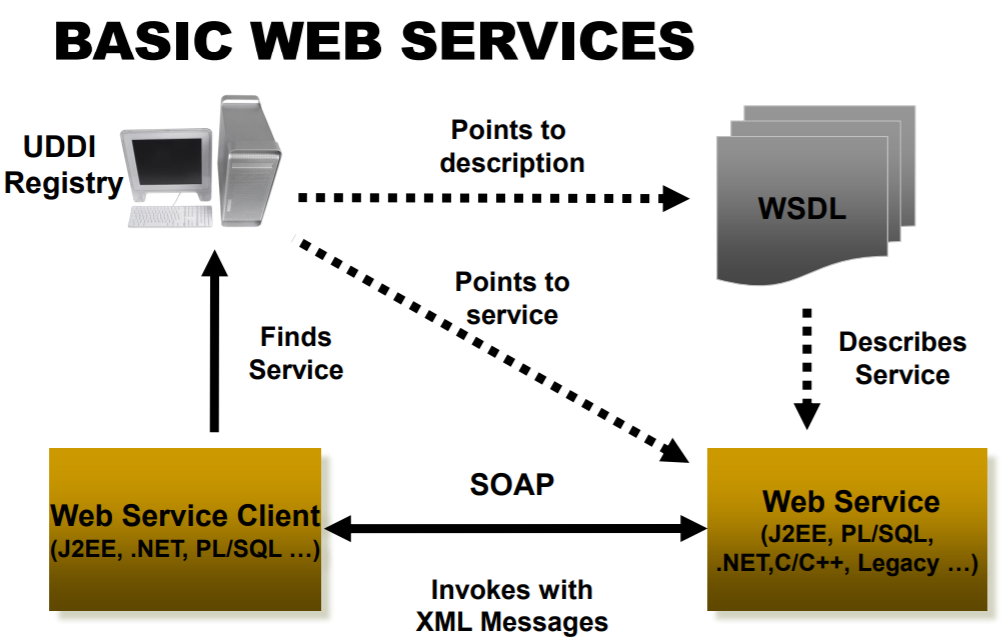
## SERVICE-ORIENTED ARCHITECTURE (SOA)

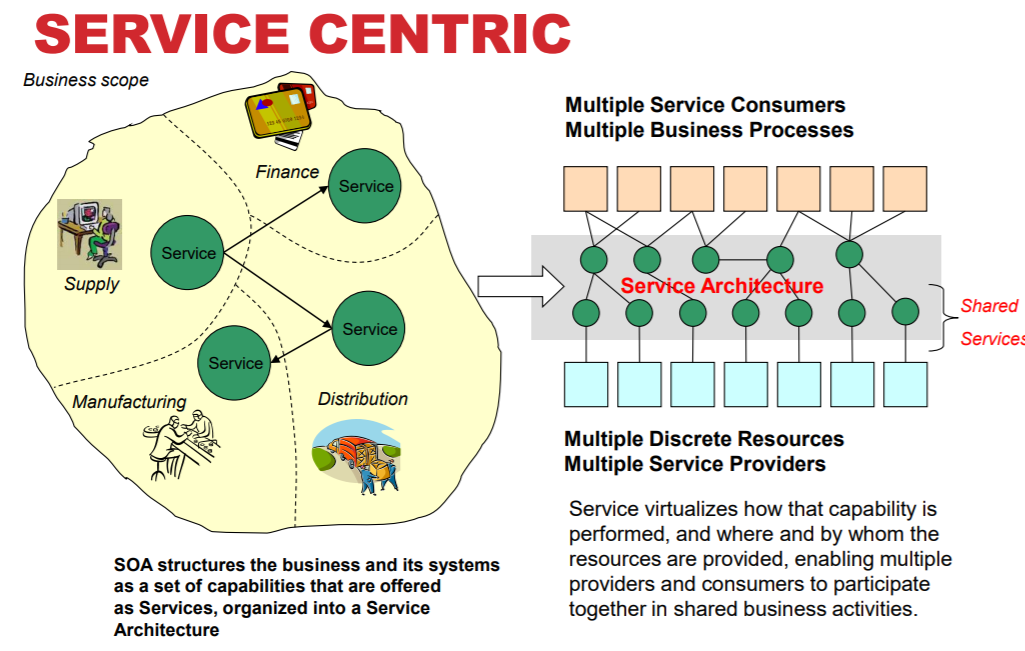
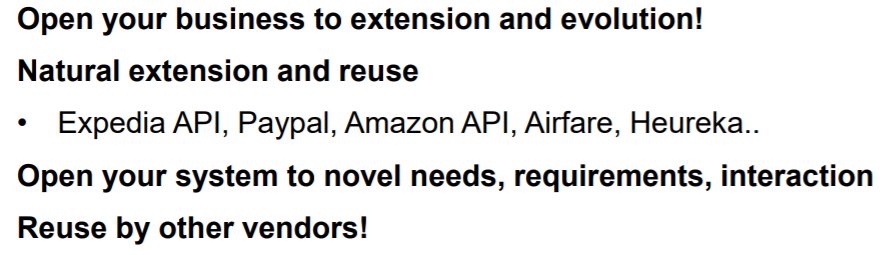
* So far we considered that server-side app offers data, knowledge and presentation
* Service does not provide presentation
* Well accepted format: JSON, SOAP, XML..



**Vlastnosti service**

* Loose coupling
* Reusable
* Stateless
* Autonomous (independent)
* Discoverable
* Abstract
* Composable
* Platform independent

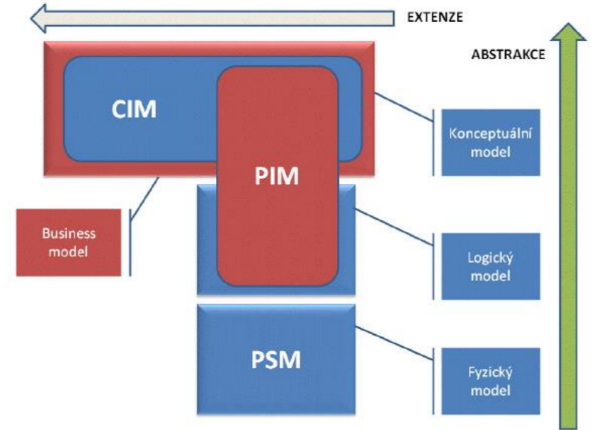


## JMS (Java Messaging Service)

* Java Messaging Services se řadí do skupiny Message Oriented Middleware (dále MOM), jiným názvem také Enterprise Messaging Systém
* Umožňují různým byznys komponentám fungovat dohromady a tvořit tak ucelený, spolehlivý a pružný systém.
* Programátor se díky JMS nemusí starat o technické pozadí za koloběhem zpráv
* Výhody :
  + vyšší pružnost
  + podpora Java, J2EE servers
* Mechanismy
  + asynchronní point-to-point / synchronní point-to-point
  + Publish / Subscribe
* **Topic** – zpráva je doručena všem, kteří se zaregistrují k její konzumaci
* **Queue** – zpráva je doručena pouze jednomu konzumentovi. Po přečtení se z fronty vymaže.
* Použití: Například JMS pro odesílání mailů (Queue). JMS pro notifikaci o změně zrušení události (Topic).

## MDA – Model driven architecture

* Založené na standardech OMG
* Hlavní myšlenkou je oddělit business a aplikační logiku od technologické platformy
* Přináší postupy, jak tyto modely správně transformovat.
* Primárními cíli tohoto přístupu jsou zajištění přenosnosti, interoperability (součinnosti) a znovu použitelnosti díky oddělené architektuře.
* Existuje model i kód vedle sebe (náročné na udržovatelnost)
* Př.: jazyk ATL = Alternativní transformační jazyk
  + Probíhají transformace mezi modely

**Rozčlenění architektury**

* Model nezávislý na počítačovém zpracování (CIM)
* Model nezávislý na platformě (PIM)
* Model specifický na konkrétní platformě (PSM)
* Zdrojový kód aplikace (výsledná implementace)

**(CIM) Computation Independent Model - Requirements**

* Je určen pro analytika
* Model, též známý jako [doménový model](https://cs.wikipedia.org/w/index.php?title=Dom%C3%A9nov%C3%BD_model&action=edit&redlink=1), se zaměřuje výhradně na prostředí a obecné požadavky systému
* Tento model reflektuje „business“ požadavky zákazníka a pomáhá přesně popsat to, co se od systému očekává.
* Zásadně neřeší otázku, jak se, co implementuje, jeho hlavní cíl je popsat co má systém dělat z pohledu zadavatele.

**(PIM) Platform Independent Model – Design**

* Tento model se zabývá tou částí kompletní specifikace systému, která se nemění podle konkrétního druhu zvolené platformy.
* Popisuje chování (algoritmy) a strukturu aplikace opravdu jen v těch mezích, které zajistí jeho přenositelnost mezi různými technologickými řešeními. Oproti předcházejícímu modelu je doplněn o ty informace (algoritmy, principy, pravidla, omezení…), na různé platformy podobného typu.
* PIM zprostředkovává určitou míru nezávislosti konkrétního řešení dané problémové oblasti tak, aby se hodila

**(PSM) Platform Specific Model – Implementation**

* Poslední model (pokud se nebere v úvahu zdrojový kód) specifikace MDA, který je již závislý na cílové platformě, kombinuje PIM s konkrétním technologickým řešením (např. .NET, Java EE, Corba).
* Tento model věrně odráží strukturu kódu, a proto je již dostatečným podkladem pro implementaci

**(PDM) Platform Description Model** – Architecture design

* Popisuje technologické koncepty určité platformy, které mohou být využity při realizace aplikace. Př.: CORBA Component Model.

## CORBA - Common Object Request Broker Architecture

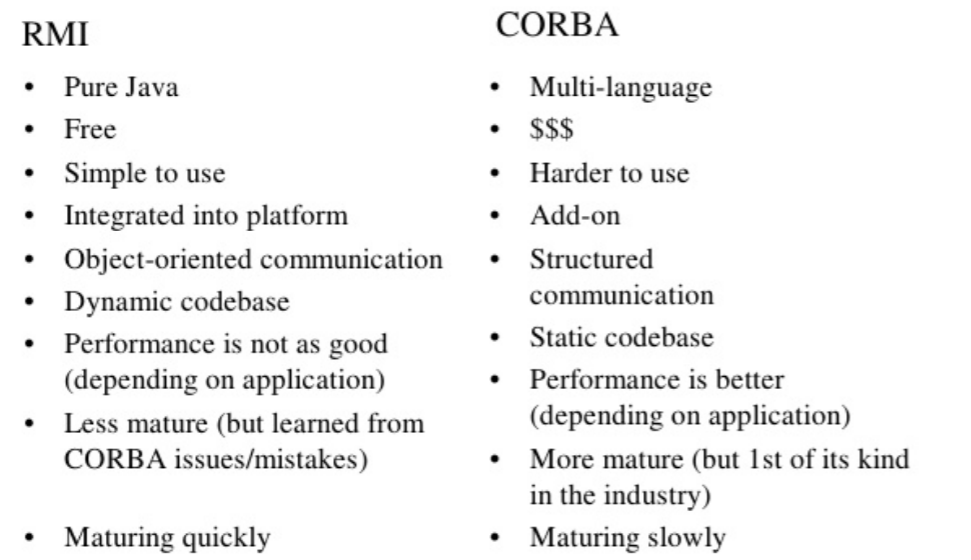
* standard pro podporu komunikace distribuovaného systému
* pouze specifikace
* standard designed to facilitate the communication of systems that are deployed on diverse platforms.
* CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware.
* CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented.

**princip**

* Objekt poskytuje službu
* Služba je dána pomocí interface
* Interface je definován pomocí Interface Definition Language (IDL)

**definuje**:

* metody a atributy podporované objektem
* výjimky vyvolané objektem
* datové typy návratových hodnot metod



# Sdílení dat

* Sdílení dat může probíhat různě podle architektury
* client-server
* Replikace
* žádná
* částečná
* úplná
  + primary copy model
  + gossip architecture

## Architektury P2P

* označení typu počítačových sítí, ve které spolu komunikují přímo jednotliví klienti (uživatelé). Opakem je klient-server, ve které jednotliví klienti komunikují vždy s centrálním serverem či servery
* všechny uzly sítě jsou si rovnocenné
* Jednou ze základních výhod P2P sítí je fakt, že s rostoucím množstvím uživatelů celková dostupná přenosová kapacita roste

## Plná Replikace dat

* **Primary-copy** model
  + Existuje jeden primární prvek pro uchovávání dat, ke kterému přistupují klienti nebo servery
  + Tento prvek se poté zálohuje do «Replic»
* **Gossip** architektura
  + Existuje více prvků s daty, ke kterým přistupují klienti nebo servery
  + Tyto datové prvky spolu komunikují pomocí gossip zpráv (výměna, aktualizace dat)

## Koherence

Paměťová koherence (anglicky cache coherence), resp. koherence vyrovnávajících pamětí (cache) znamená jednotnost dat ve více vyrovnávacích pamětích.

Problém udržení koherence vyvstává u multiprocesorových počítačů s více cachemi. Při změně dat ve vyrovnávací paměti jednoho procesoru je nutné zabránit tomu, aby ostatní procesory (využívající jiné cache) pracovaly se zastaralými daty. Docílit shodných dat na všech pamětech je však složité.

## Konzistence

Konzistence oproti koherenci specifikuje, v jakém pořadí jednotlivé procesy spouštějí své paměťové operace, či jak se toto pořadí jeví ostatním procesům.

Konzistence definuje, co je korektní chování sdílené paměti z pohledu čtení a zápisů

# Applety

* Applet je java program upravený ke spuštění na straně klienta (v prohližeci)
* Applety se načítají nezávisle na celé stránce
* V html stránce je potřeba identifikovat místo kde bude applet: <APPLET ....> .... </APPLET>
* Prohlížeč poté stáhne bytekód programu a spustí
* Prohlížeč musí mít lokálně instalovaný JRE (není potřeba JDK)
* V jedné stránce může být i více appletů mohou mezi sebou komunikovat
* Applet nemůže škodit klientskému počítači, neboť nemůže:
  + přistupovat k lokálním souborům
  + zjišťovat a měnit vlastnosti systému
  + vytvářet knihovny a definovat nativní metody
  + spouštět programy
  + navazovat komunikaci s třetí stranou
* Je-li však applet podepsaný ( signed ), může klient tato omezení zmírnit
* Můžeme udělit grant appletu na určené operace

# Mobilita = podivné slidy

* **Mobilita**: Jedna z vlastností systému/ podsystému/ algoritmu, založena na migrování výpočtu z jednoho uzlu sítě na další, využívá k tomu « agenty »
* Důležitá u **distribuovaných** výpočtů (dost zásadní)
* Technologie: ●
  + Voyager - Object Space - JVM
  + Odyssey - General Magic - JVM

**Agenti**

* Program, který se chová jako jiný uživatel
* Nepotřebuje dohled, reagují na prostředí
* Někdy označované jako «bots »
* Využití pro umělou inteligenci
* CO dělají
  + Provádět cílené vyhledávání na internetu.
  + Zkontrolovat a upřednostněte příchozí e-maily.
  + Otestovat nové počítačové hry.
  + Vyplňovat e-formuláře.
  + Provádět online hledání práce.
  + Synchronizovat profilů sociálních sítí.
  + Sestavit přizpůsobené zprávy.
  + Najít dobré nabídky v e-commerce.

**typy**

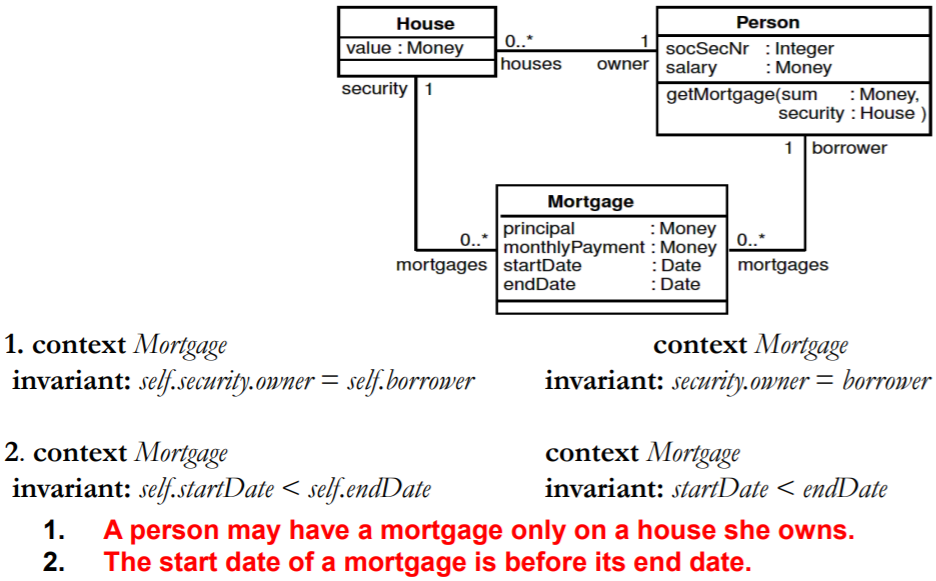
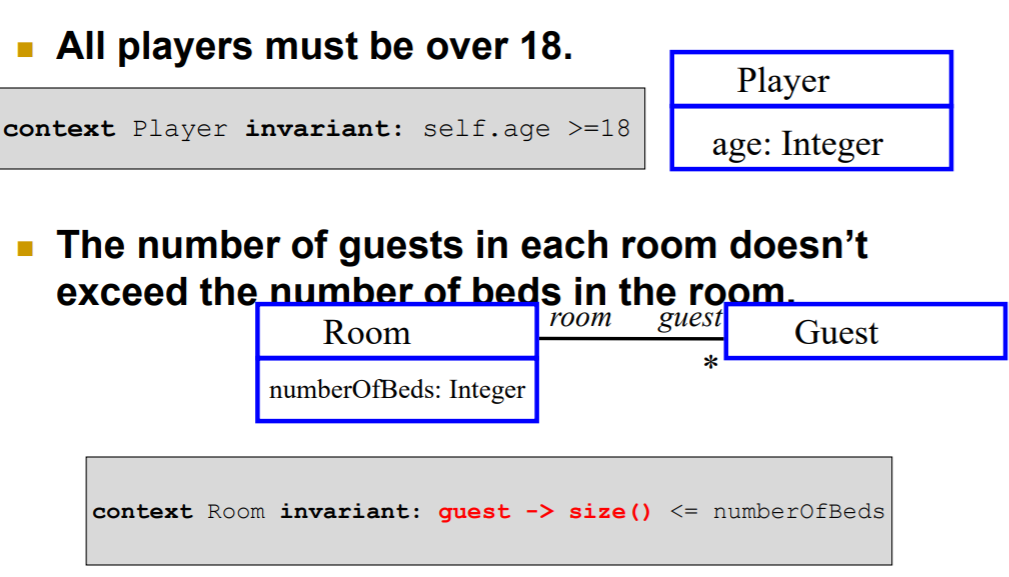
* Autonomní agenti: AI – modifikace chování pro dosažení optima
* Multiagentní systémy: AI – kooperace jinak autonomních agentů
* Distribuovaní agenti: spolupráce komponent výpočtu v distribuovaném prostředí
* Mobilní agenti: distribuovaní agenti s možností migrace

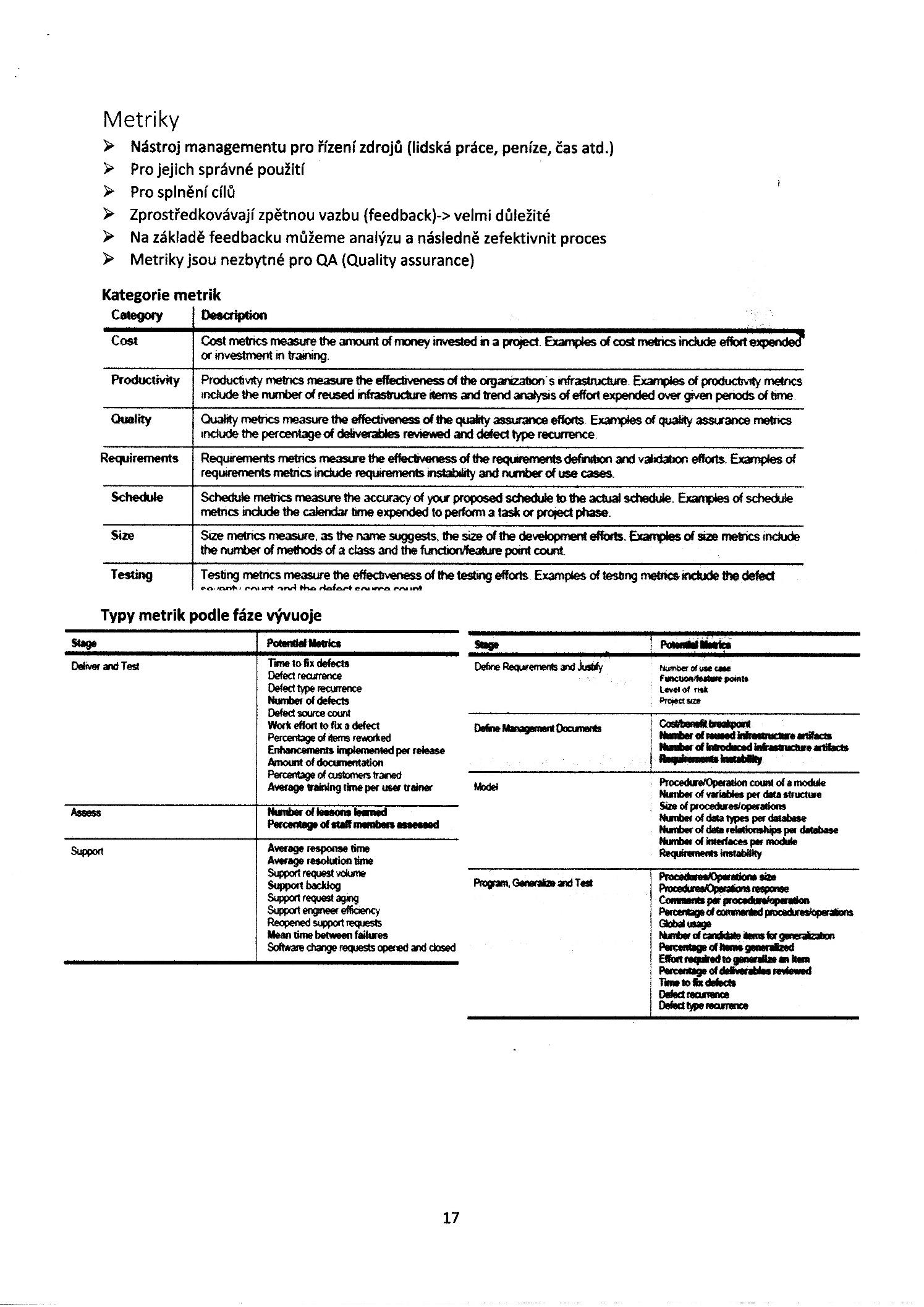
# OCL

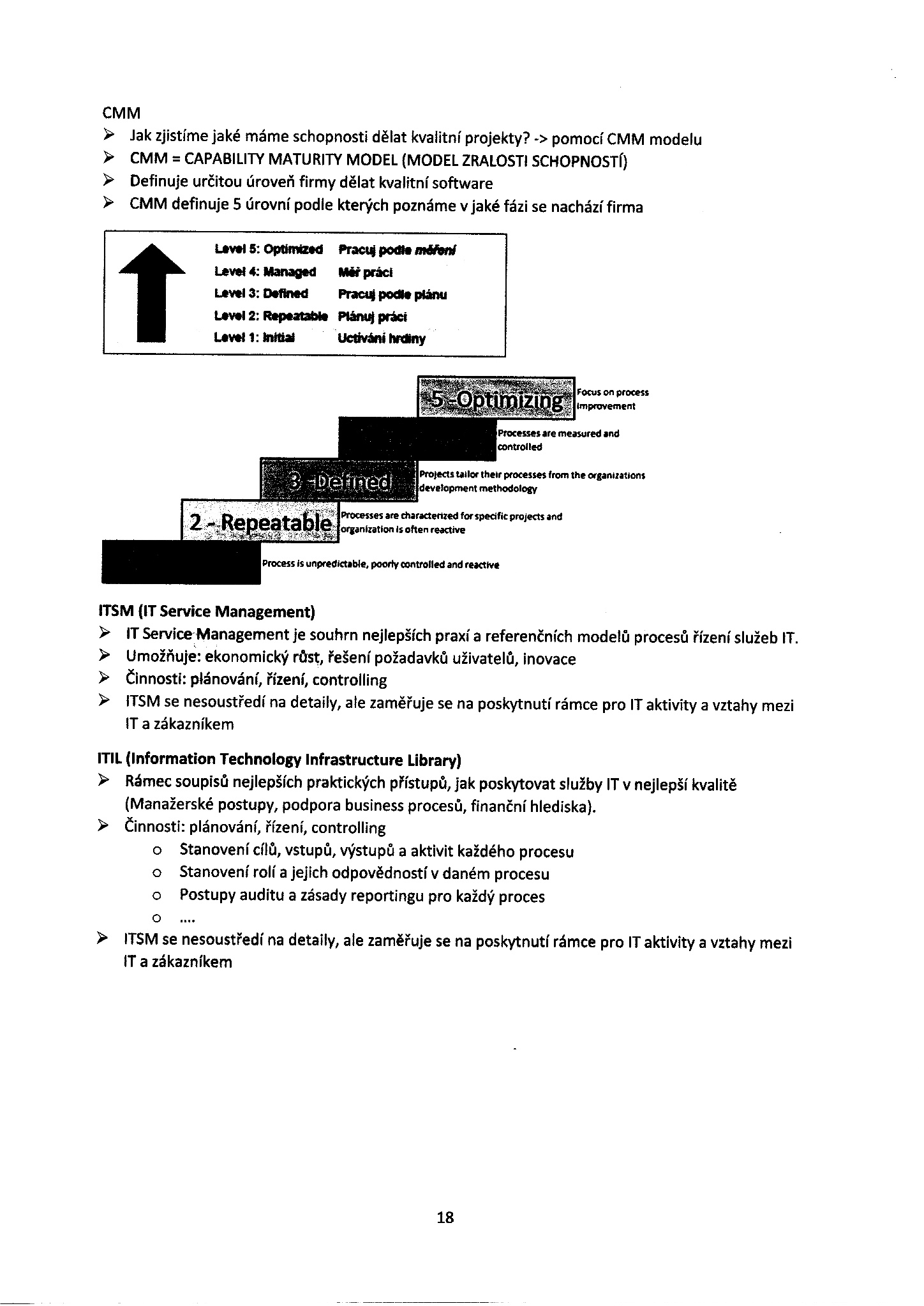
Jazyk Object Constraint Language (OCL) poskytuje rámec pro specifikaci omezení kladených na model formálním způsobem.

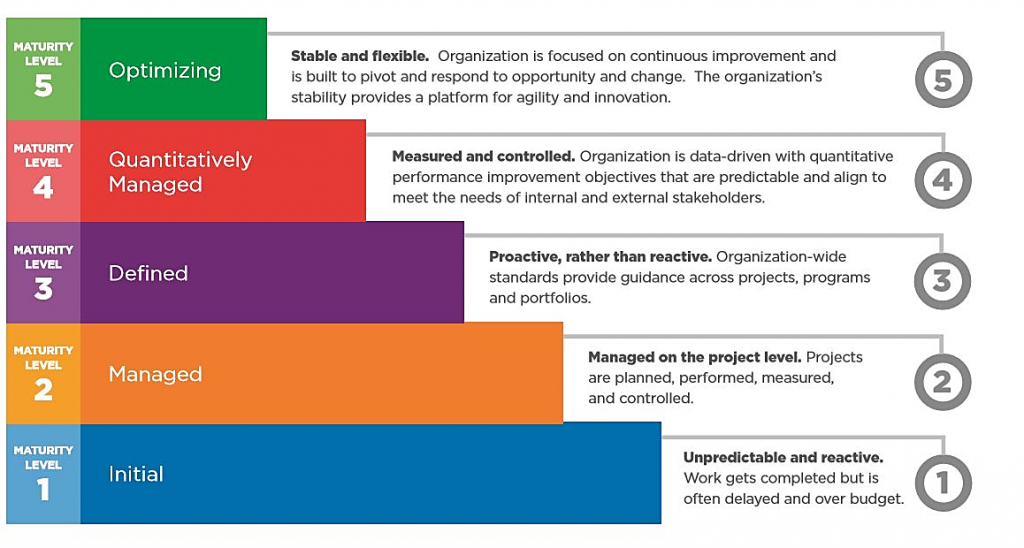
Specifikace invariantů tříd a typů

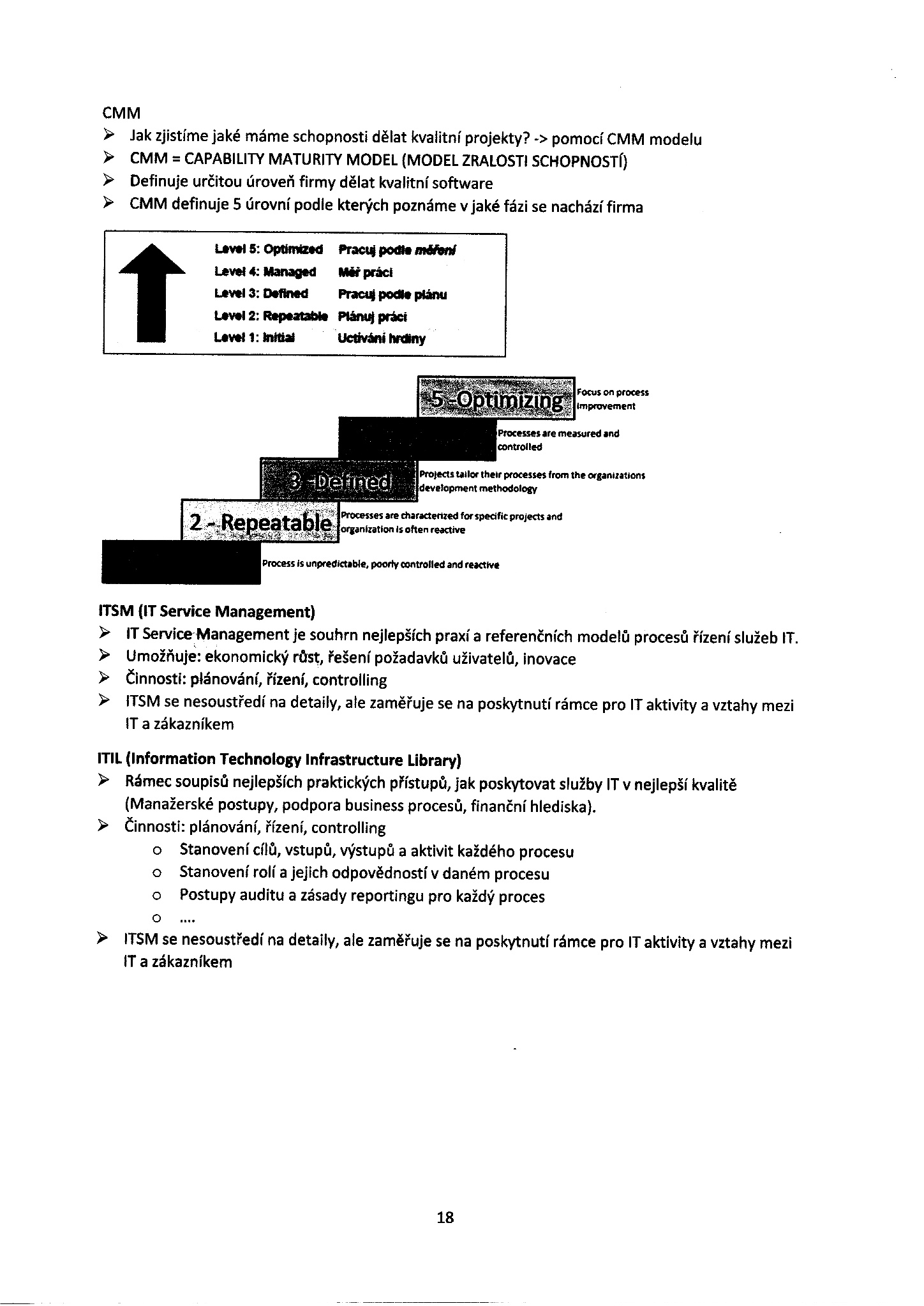
* Vymezení legálních hodnot
* **Pre** a **post**-conditions pro metody
* Navigační jazyk
* Omezení/vymezení operací











# Pojmy

* **Cohesion** - how closely related everything is with one another.
* **Coupling** - how everything is connected to one another.
* **Funkční požadavky**: jaké všechny funkcionality z pohledu uživatele má systém vykonávat.
* **Nefunkční požadavky**: Nefunkční požadavky softwarové architektury jsou požadavky, které kladou omezení na design a provedení, například rychlost odezvy

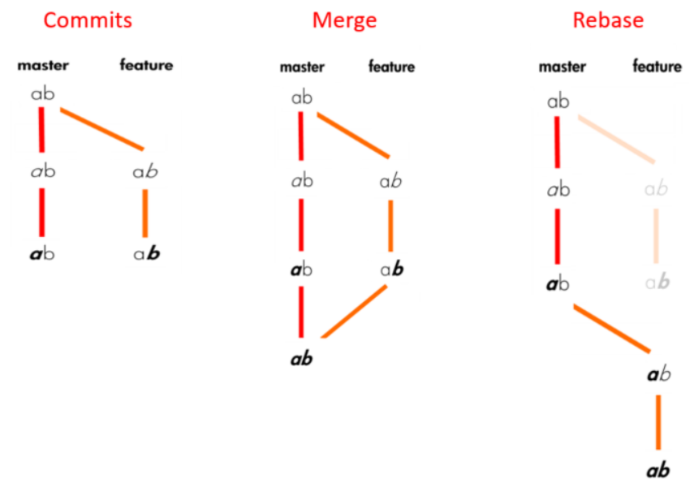
# Kafka

* Apache Kafka se používá pro budování real-time datových přenosů a streamingových aplikací.
* Každý záznam se skládá z klíče, hodnoty a časového razítka.
* Kafka má API čtyři **klíčové** funkční části:
* **Producer** API umožňuje aplikacím publikovat streamovací záznamy do jednoho nebo více Kafka topiců
* **Consumer** **API** umožňuje aplikacím se přihlásit k odběru jednoho nebo více topiců a zpracovat stream záznamů do nich plynoucí
* **Streams** **API** umožňuje aplikacím pracovat jako stream procesor, konzumovat vstupní stream z jednoho nebo více topiců a tvořit výstupní stream do jednoho nebo více výstupních topiců, a přitom efektivně transformovat vstupní streamy na výstupní
* **Connector API** dovoluje tvořit a spouštět opakovatelné producery a consumery, které propojují Kafka topicy s existujícími aplikacemi nebo datovými systémy. Například, konektor do relační databázi může zachycovat každou změnu do tabulky.

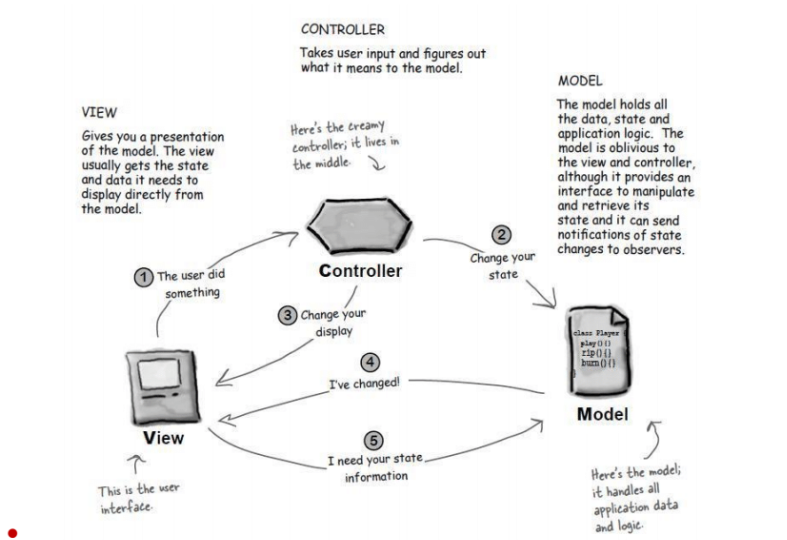
# Třívrstvá architektura

* Prezentační vrstva - zobrazuje informace pro uživatele (grafické uživatelské rozhraní), může kontrolovat zadávané vstupy, neobsahuje zpracování dat
* Aplikační vrstva (Business Logic) - jádro aplikace, její logika a funkce, výpočty a zpracování dat
* Datová vrstva - databáze, která data uchovává, zpřístupňuje a zaručuje jejich konzistenci

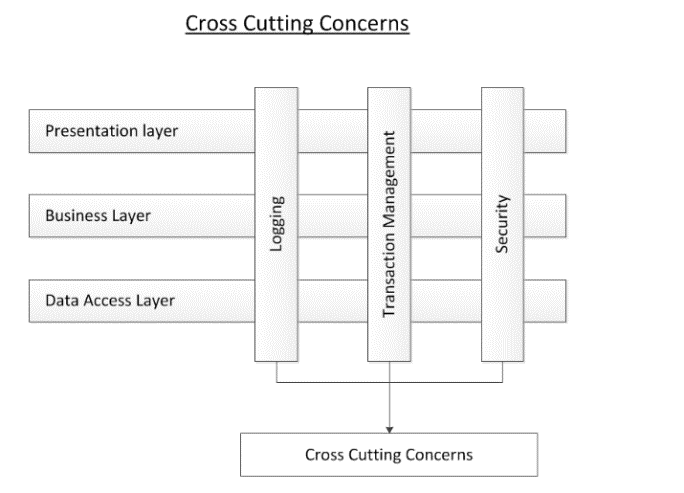
# Git

**Git rebase** in standard mode will automatically take the commits in your current working branch and apply them to the head of the passed branch

# MVC



odděluje uživatelská rozhraní od své logiky a od dat

* **Model - reprezentuje** znalosti, zachycuje systémová data, notifikování (informovat o změnách), observer pattern, doménově specifická reprezentace informací, s nimiž aplikace pracuje
* **View - vizuální** reprezentace modelu převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli
* **Controller - spojení** mezi uživatelem a systémem, poskytuje uživateli vstup, menu, příkazy, bere vstupy od uživatele, reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu

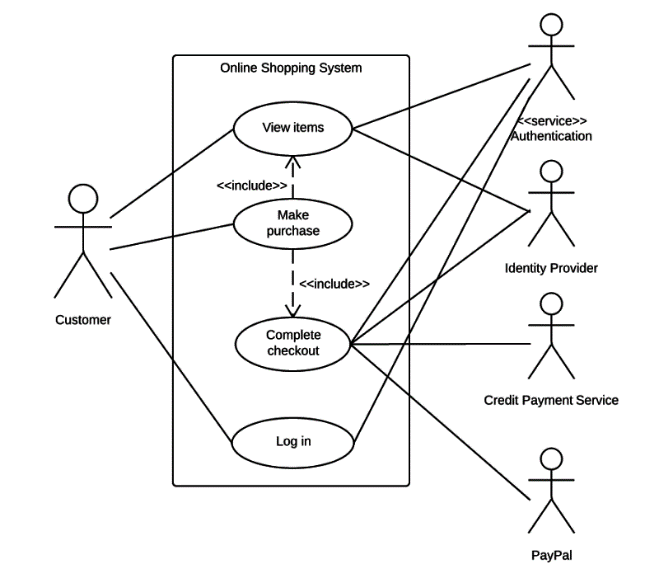
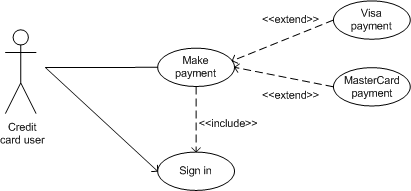
# AOP

In computing, aspect-oriented programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It extends OOP.

# Softwarové inženýrství

Softwarové inženýrství je činnost zahrnující inženýrství, informatiku a management, jejímž cílem je návrh, tvorba a údržba počítačových programů.

Usecase diagram



# UML

