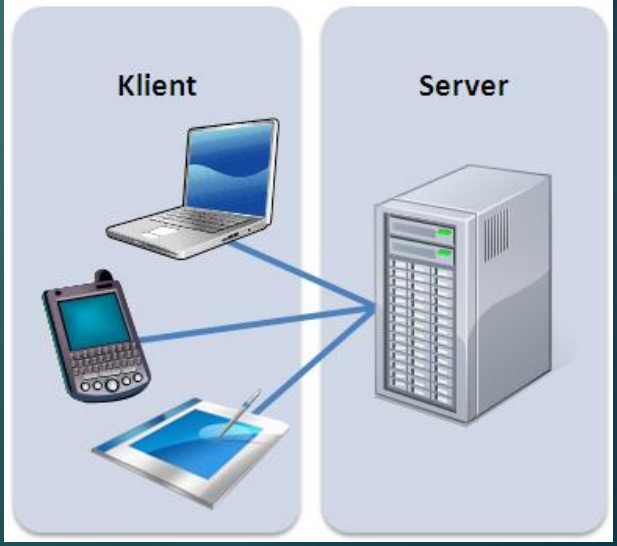
Další důležité přístupy k architektuře IS/IT

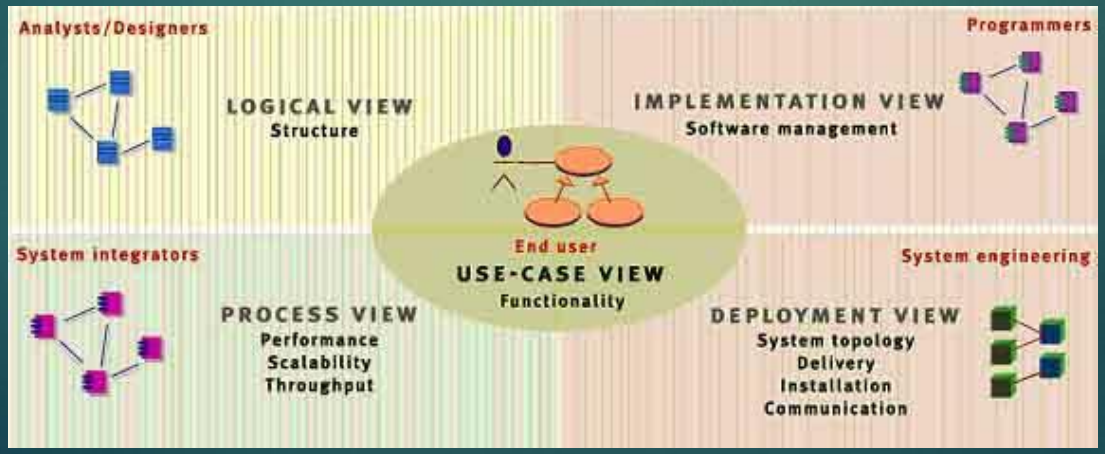
- Architektura 4+1 pohledů
- Komponentová architektura
- Servisně orientovaná architektura
- Modelem řízená architektura (MDA)
- Architektonický vzor MVC, MVVM, MVP

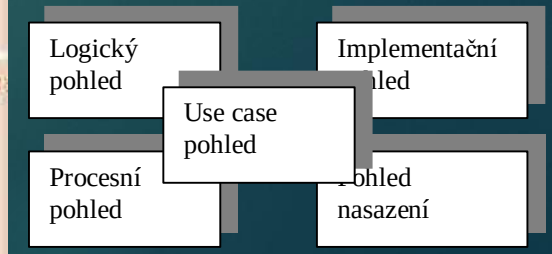


Architektura 4+1 pohledů

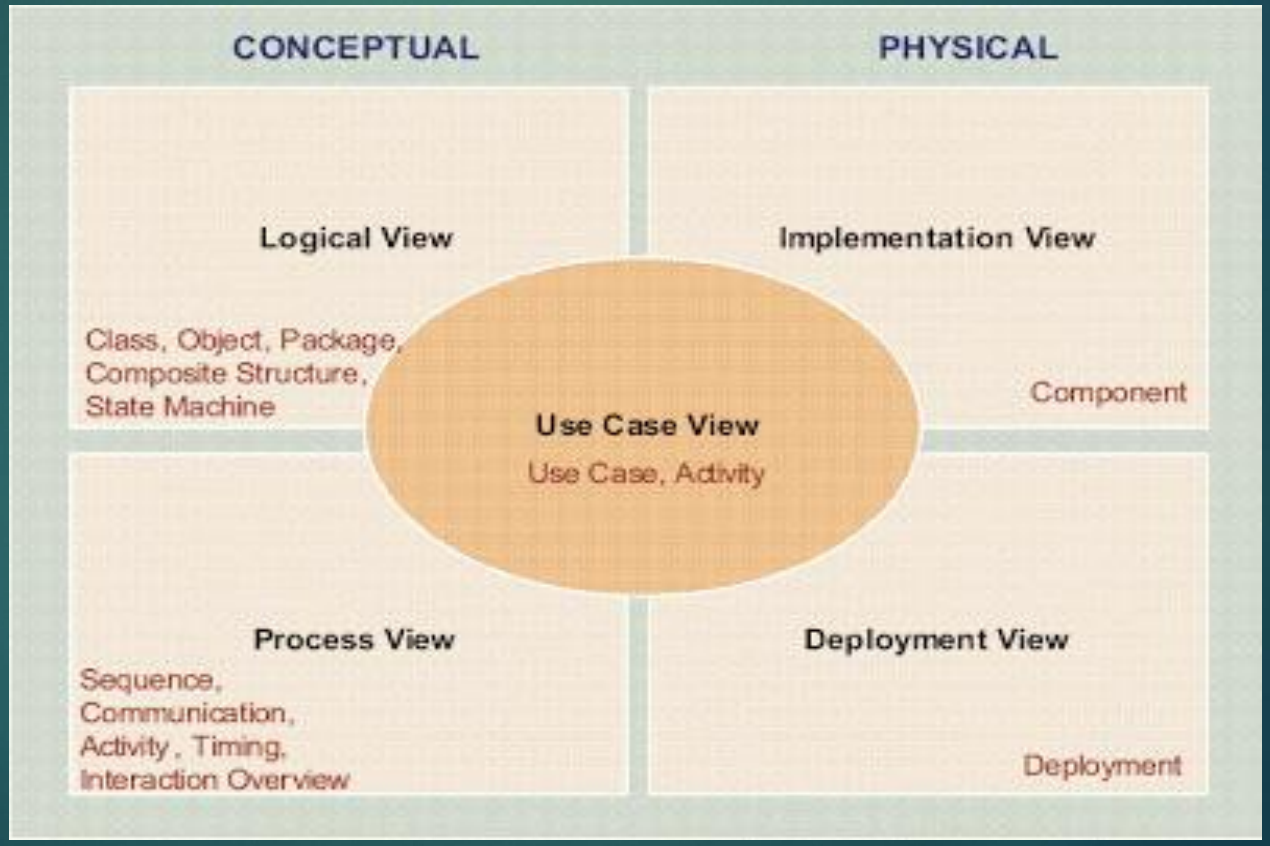
Definice architektury systému podle IEEE: nejvyšší úroveň koncepce systému v jeho vlastním prostředí.

- Architektura systému musí mít schopnost být viděna z různého úhlu pohledu.
- 4+1 pohledy (každý pohled odpovídá modelu v dokumentaci systému).
- Spojena s objektovým přístupem.

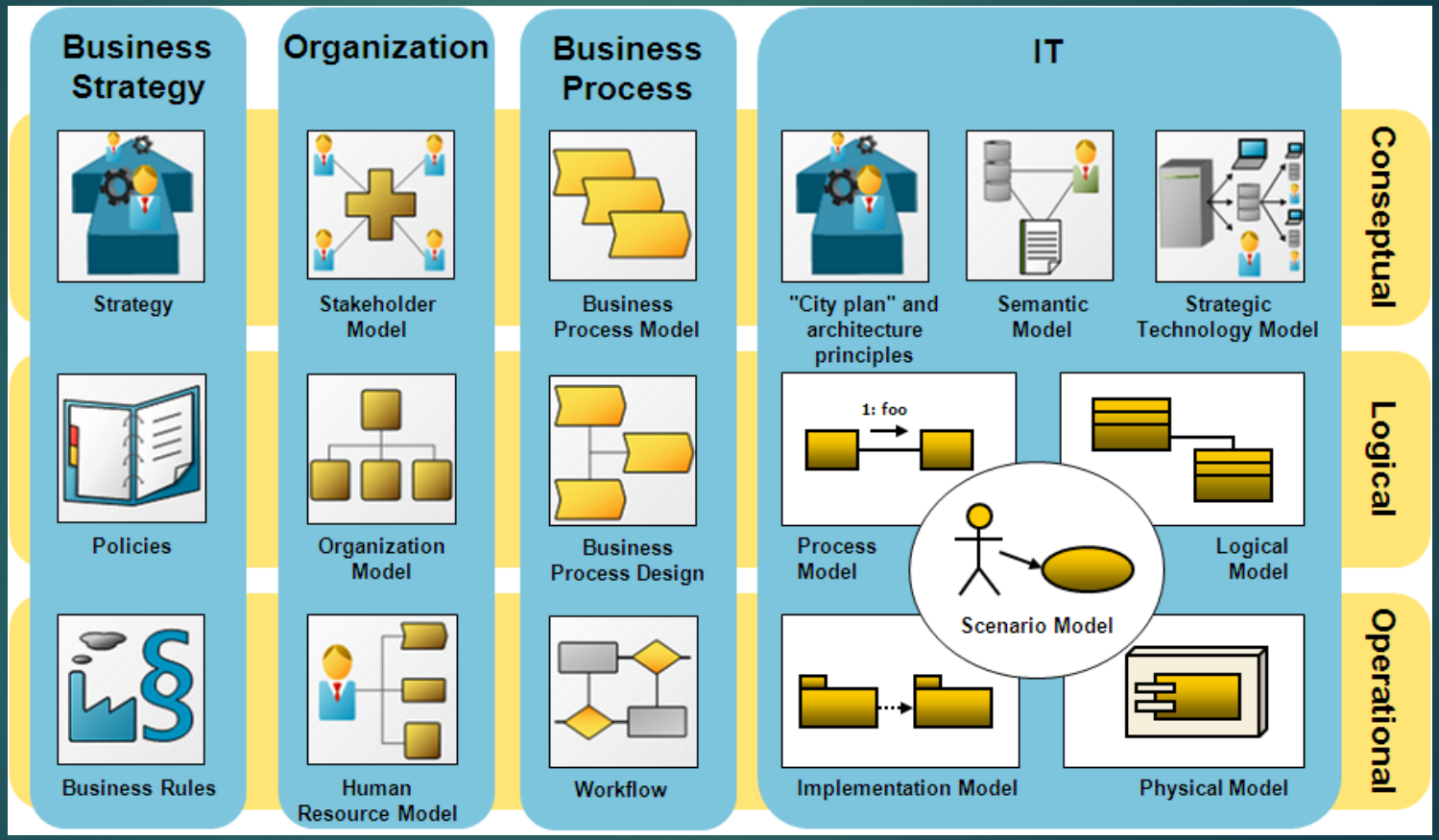




4+1 a UML diagramy



Pozn. 4+1 a vrstvy v podniku



Architektura 4+1 pohledů

Logický pohled (slovníček, funkce)

- logická struktura systému z hlediska výsledné funkčnosti (co by systém měl vykonávat),
- identifikuje hlavní (věcné) balíky, subsystémy, třídy a vazby mezi nimi,
- model **perzistentních** informací (data i funkce),
- zajímá analytika a designera,
- zachycuje slovník oblasti problému jako množinu tříd a objektů,
- **spolu s procesním pohledem představují chování systému.**

Architektura 4+1 pohledů

Implementační pohled (vývojový pohled, systémová seskupení, správa konfigurace)

- popis organizace statických softwarových komponent, modulů (exe, html, dll) ve vývojovém prostředí,
- rozčlenění do vertikálních a horizontálních vrstev,
- zaměřen na dekompozici systému do menších samostatně realizovatelných komponent,
- zajímá programátory a SW management,
- slouží ke znázornění závislostí mezi komponentami a toho, jak konfigurovat spojení těchto komponent.

Architektura 4+1 pohledů

Procesní pohled (výkon, škálovatelnost, dostupnost, propustnost)

- pohled na chování systému,
- konkurence a paralelismus, propustnost,
- automatizované úlohy, tolerance chyb,
- zotavení z běhových chyb,
- odezva systému na vnější podněty,
- rozšiřitelnost systému, jeho výkonnost, přípustnost,
- zajímá systémové integrátory.

Architektura 4+1 pohledů

Pohled nasazení (pohled fyzický, technologie systému, distribuce, doručení, instalace)

- navázání systému na topologii hardwarových a dalších softwarových komponent (jak jsou spouštěče a ostatní běhové komponenty mapovány do základních platforem a počítačových uzlů),
- fyzické rozložení komponent, nasazení, instalace, ladění výkonu,
- zajímá všechny tvůrce systému.

Architektura 4+1 pohledů

Pohled případů užití (základní požadavky na systém)

- speciální role, obsahuje klíčové případy užití (use case),
- pomáhá odhalit základní požadavky na architekturu,
- zpětná vazba na věcné požadavky při testování,
- systematický přístup k věcným požadavkům,
- zajímá koncové uživatele i vývojáře.

Komponentová architektura

- Základní myšlenkou komponentových architektur je snaha pojmout tvorbu programu jako **skládání** jinak do značné míry **nezávislých komponent**. Tuto myšlenku lze uplatnit jak na úrovni **návrhu architektury**, tak na úrovni **implementace programu**.
- Na úrovni **návrhu architektury** je předním přínosem komponent možnost rozdělit řešení do **menších, a tak snáze zvládnutelných částí**. Při důsledném použití hierarchické dekompozice se pak návrh architektury na nejvyšší úrovni skládá z relativně malého počtu spolupracujících komponent.

Pozn.:
Běžně používané jednotky dekompozice jako moduly či třídy jsou zatíženy **vazbami na prostředí** pro implementaci programu, ve kterém mají další, s dekompozicí nesouvisející, funkce.
Komponenty takové vazby nemají, a proto jsou jako koncepty pro dekompozici vhodnější.

Komponentová architektura

- Na **úrovni implementace** programu je nejviditelnějším rysem komponent podpora **opakovaného použití** již implementovaných komponent ve větším počtu programů. Takové opakované použití slibuje mnoho výhod, mezi jinými např.:
  - zvýšení produktivity, neboť opakovaně použitý kód je možné převzít a není třeba vyvíjet nový,
  - zvýšení kvality programů, neboť opakovaně použitý kód je častěji zkoušen a zjištěné chyby jsou současně odstraněny ve všech programech, ve kterých je kód použit,
  - zvýšení výkonu programů, neboť opakovaně použitý kód se spíše vyplatí optimalizovat a každá optimalizace se současně projeví ve všech programech, ve kterých je kód použit,
  - zvýšení schopnosti programů spolupracovat, neboť opakované použití kódu vede k použití stejných formátů dat ve všech programech, ve kterých je kód použit.

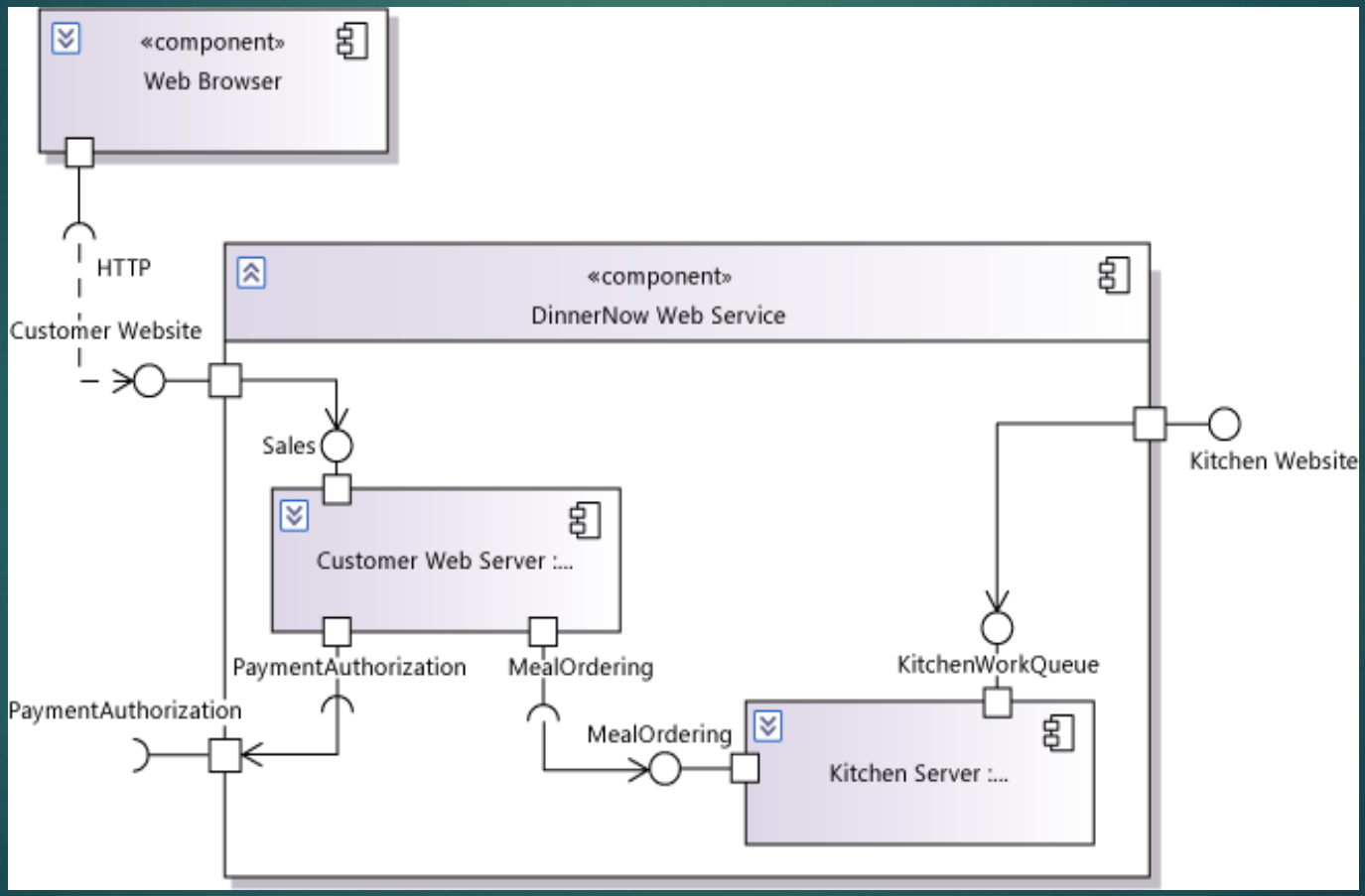
Komponentová architektura

- Komponenta = základní jednotka použitelnosti
- Je to netriviální, téměř nezávislá, vyměnitelná část systému, která poskytuje jasnou funkcionalitu. Komponenta fyzicky implementuje sadu rozhraní a musí splňovat jimi definované chování.
- Rozlišujeme:
  - spustitelné komponenty, systémové komponenty, vývojářské komponenty,
  - věcné komponenty (business component),
  - univerzální a další komponenty.

Příklady komponent (spustitelný program, knihovna, textový soubor, tabulka v databázi, ...).

Komponenty vznikají při tvorbě architektury jako samostatně navržené, vyvinuté a otestované části softwaru, integrované do systému. Existují znovupoužitelné komponenty pro řešení častých problémů, vyvinuté v rámci projektu nebo samostatně (standardy JavaBeans, CORBA, ActiveX).

Komponentová architektura - ukázka



Komponenty a vazba na softwarové architektury

- Komponentové systémy (Component-Based Systems) jsou konkrétní realizací softwarových architektur, kdy:
- **Moduly** = black-box komponenty komunikující výhradně skrz publikovaná rozhraní
- **Konektory** = spoje a kanály zajišťující komunikaci a spolupráci zkompilovaných modulů (komponent) implementovaných často odlišnými prostředky a nezávisle na sobě
- **Nasazení** = mapování modulů a konektorů na hardwarové (nebo softwarové) zdroje

Např. síťové linky, fyzické zdroje a servery v případě hardwarových zdrojů

Např. komponentový framework či middleware v případě softwarových zdrojů

Komponenta není třída

Komponenta (vs. objekt/třída):

- Spustitelná běhová jednotka
- Definovaná rozhraními (IDL)
- Typicky složena z více tříd/objektů (hierarchicky)
- Bez viditelného zdrojového kódu (black-box/grey-box)
- Vyvinuta odděleně od ostatních komponent (pro znovupoužitelnost) – propojení výhradně přes rozhraní, bez přímých referencí (na rozdíl od balíků apod.)
- Kompilaci předchází zasazení do kontextu
- Závislost na běhovém prostředí (middlewaru)

Výhody komponentové architektury

- Přehledná architektura
- Rozdělení odpovědnosti
- Komunikace pomocí rozhraní
- Řešení komplexnosti
- Zrychlení vývoje
- Sestavování systémů z nakoupených nebo dříve vyrobených a otestovaných komponent
- Oddělený vývoj balíků komponent
- Samostatné testování komponent
- Provádění integračních testů

Architektura orientovaná na služby (SOA)

- Moderní přístup k řešení architektury IS/IT, který využívá všech silných stránek předchozích přístupů a přidává koncept **opakovatelně používaných komponent – služeb** s přesně definovaným chováním a rozhraními na okolí, tj. webových služeb.
- Zavádí se pojem **katalog služeb** (seznam všech služeb, které jsou v daném řešení k dispozici a je možné je kombinovat).
- Hlavním rysem je **flexibilita při návrhu konkrétního řešení**, tj. aby při změně obchodního požadavku na fungování IS stačilo v katalogu najít služby a jejich kombinací rychle dodat uživatelům požadovanou funkcionalitu. Komponenty lze z katalogu vyvolat opakovaně – snížení nákladů.
- Viz obr.

Základní principy SOA

- Standardizovaný kontrakt služeb (Standardized Service Contract) daný smlouvou SLA (Service Level Agreement)
- Volné spojení (Loose coupling)
- Abstrakce služby
- Přepoužitelnost služby
- Autonomie služby
- Služba přes příslušnosti
- Zjistitelnost služby
- Rozložitelnost služby
- Interoperabilita služby

Základní principy SOA

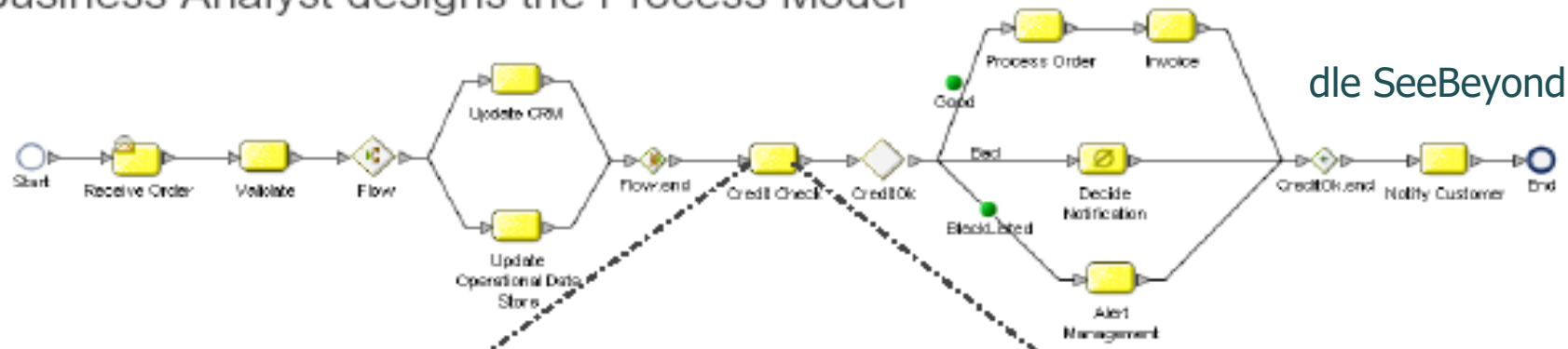
- SOA rozlišuje tzv. **poskytovatele služby** (Service Provider) a **konzumenta služby** (Service Consumer). Jednotlivé služby jsou do jisté míry samostatné celky, které mohou být obvykle vyvíjeny, provozovány či nahrazeny nezávisle na ostatních službách systému.

- Služby v tomto pojetí představují určité **stavební kameny**, pomocí nichž lze vytvářet složitější funkční celky a které (samostatně) provádí definovanou činnost, přičemž **každá služba má jednoznačně definované rozhraní, pomocí kterého komunikuje s ostatními komponentami informačního systému**. Nezabývá se otázkou vnitřní implementace jednotlivých služeb, je to uzavřený celek, u kterého známe pouze jeho rozhraní, nikoliv jeho vnitřní strukturu.

- Z technického hlediska bývají služby nejčastěji implementovány jako **webové služby (Web Service WS)**, nicméně existují i další způsoby realizace. Obecně lze servisně orientovanou architekturu a služby charakterizovat těmito pojmy jako je **modularita** (modularity), **znovu použitelnost** (reusability), **volné vazby** (loose coupling), **interoperabilita** (interoperability).

Architektura orientovaná na služby SOA

1) Business Analyst designs the Process Model

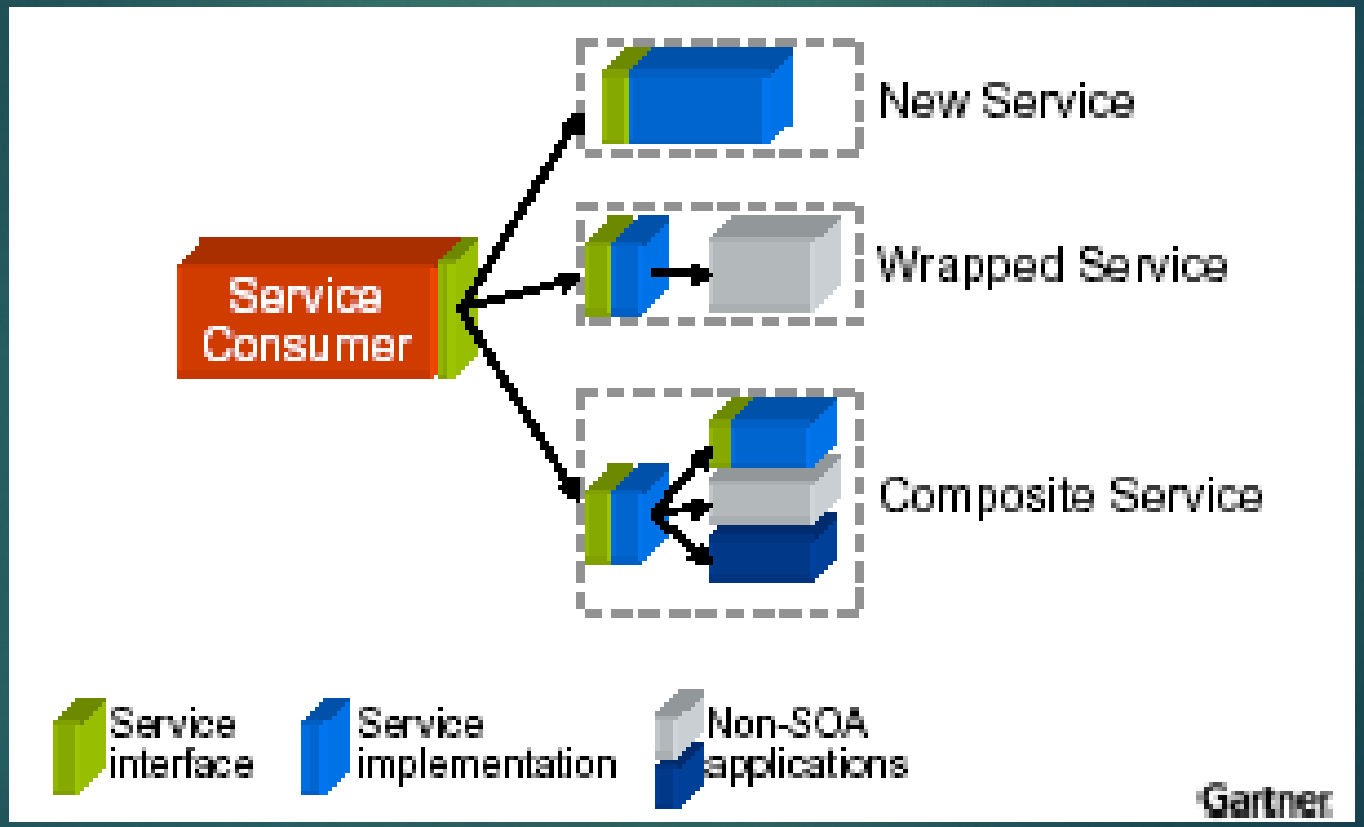


dle SeeBeyond

2) And defines the Service Interfaces – NOT THE IMPLEMENTATION

Architektura orientovaná na služby

varianty realizace služby dle Gartner



Architektura orientovaná na služby (SOA)

SOA patří mezi SW architektury

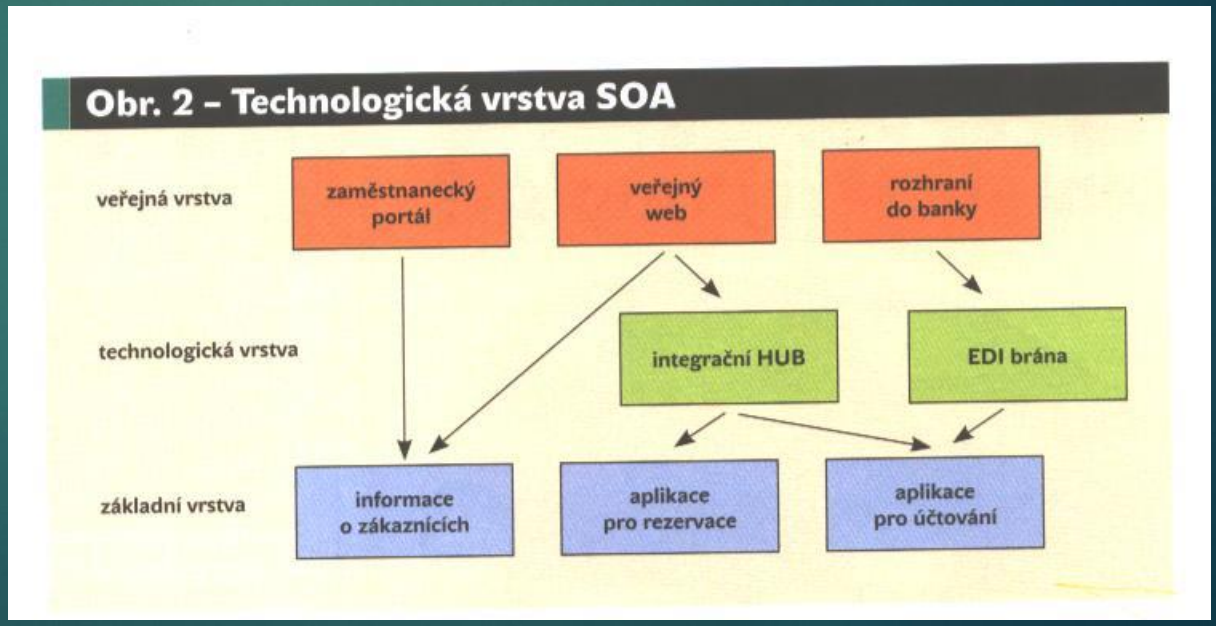
Implementace SOA

- **1. krok** : Definice základních stavebních kamenů – služeb. Služby ze základní vrstvy jsou pak zpřístupněny uživatelům přes vrstvu tzv. veřejných služeb (bankomaty, veřejný portál, SMS brána), tedy služby GUI nebo B2B rozhraní.

Architektura orientovaná na služby (SOA)

Implementace SOA

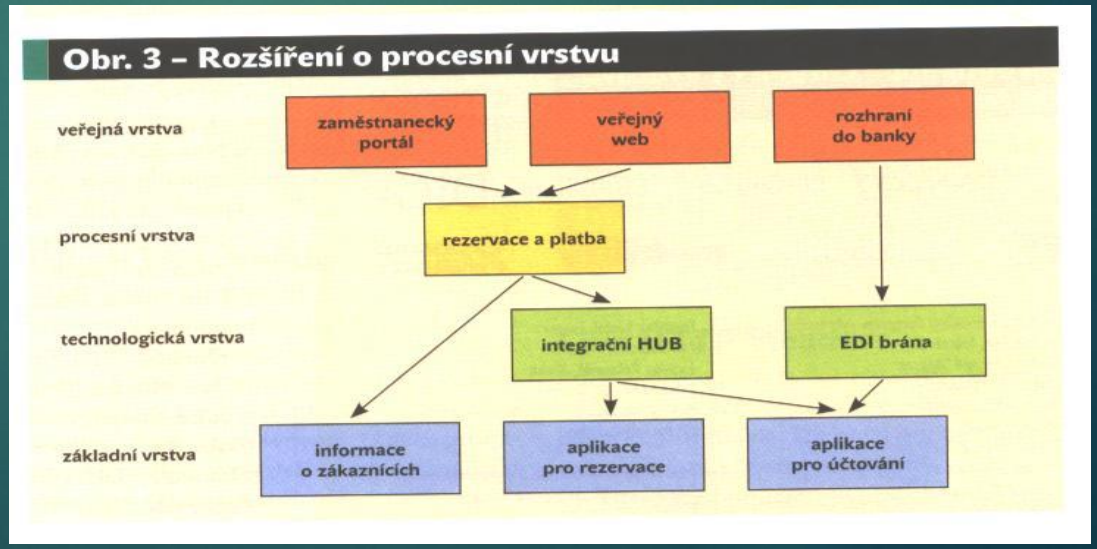
- **2. krok**: Implementace vrstvy technologických služeb. Integrační HUB zde zajišťuje výměnu dat mezi rezervačním a finančním systémem. Takto navržená architektura může zvýšit výkon celého systému oddělením integračních a prezentačních služeb.



Architektura orientovaná na služby (SOA)

Implementace SOA

- **3. krok**: Procesní vrstva umožňuje snadnou implementaci složitých podnikových procesů, které v sobě mohou zahrnovat složitá rozhodovací kritéria nebo podmíněné spouštění nejrůznějších podsystémů a aplikací. Nová služba v tomto příkladu může obsahovat různé podmínky (nabídka slev na letenky), nebo verifikaci platby s bankou.



SOA na závěr

- ERP systémy postavené na SOA se skládají z provázaných procesů, které jsou postaveny na službách a jako takové spolu komunikují. SOA vytváří generické služby jako základní stavební bloky aplikací, které pak spolu mohou interagovat napříč různými platformami. Tato architektura přináší **nezávislost na platformě, opakovanou použitelnost jednotlivých služeb a větší flexibilitu systému**.

- Koncept SOA spočívá v rozdělení jednotlivých modulů ERP na menší komponenty, které jsou opatřeny vnější vrstvou otevřeného a zdokumentovaného rozhraní. Takové rozhraní pak umožňuje komunikaci komponenty s jinou komponentou, a **to i od jiného výrobce**.

- Budoucí inovace firemního procesu se tak stává mnohem snazší, protože již není třeba implementovat celý nový modul informačního systému, ale postačí pouze dílčí změna v oblasti několika komponent (služeb). Zákazník navíc není vázán na jednoho výrobce ERP systému, ale může si vybrat z nabídky vhodných komponent od více dodavatelů, kteří podporují koncepci SOA.

Architektura založená na modelech (MDA)

- MDA je přístup společnosti OMG Object Management Group, který vychází z myšlenky, že změny v oblasti SW resp. podniku se liší v závislosti na úrovni abstrakce. Na nižších úrovních jsou změny častější a zásadnější. Z toho vyplývá, že dopady neustálých změn v technologiích a společnosti vůbec je možné omezit jen na nižší vrstvy modelu a v případě potřeby přepracovat jen ty části, kterých se změna dotýká.

- Původně bylo toto paradigma vytvořeno pro vývoj software a byly popsány tři modely: **Počítačově nezávislý model** (Computer Independent Model, CIM), **Platformově nezávislý model** (Platform Independent Model, PIM) a **Platformově specifický model** (Platform Specific Model, PSM) se vztahem ke komponentní technologii.

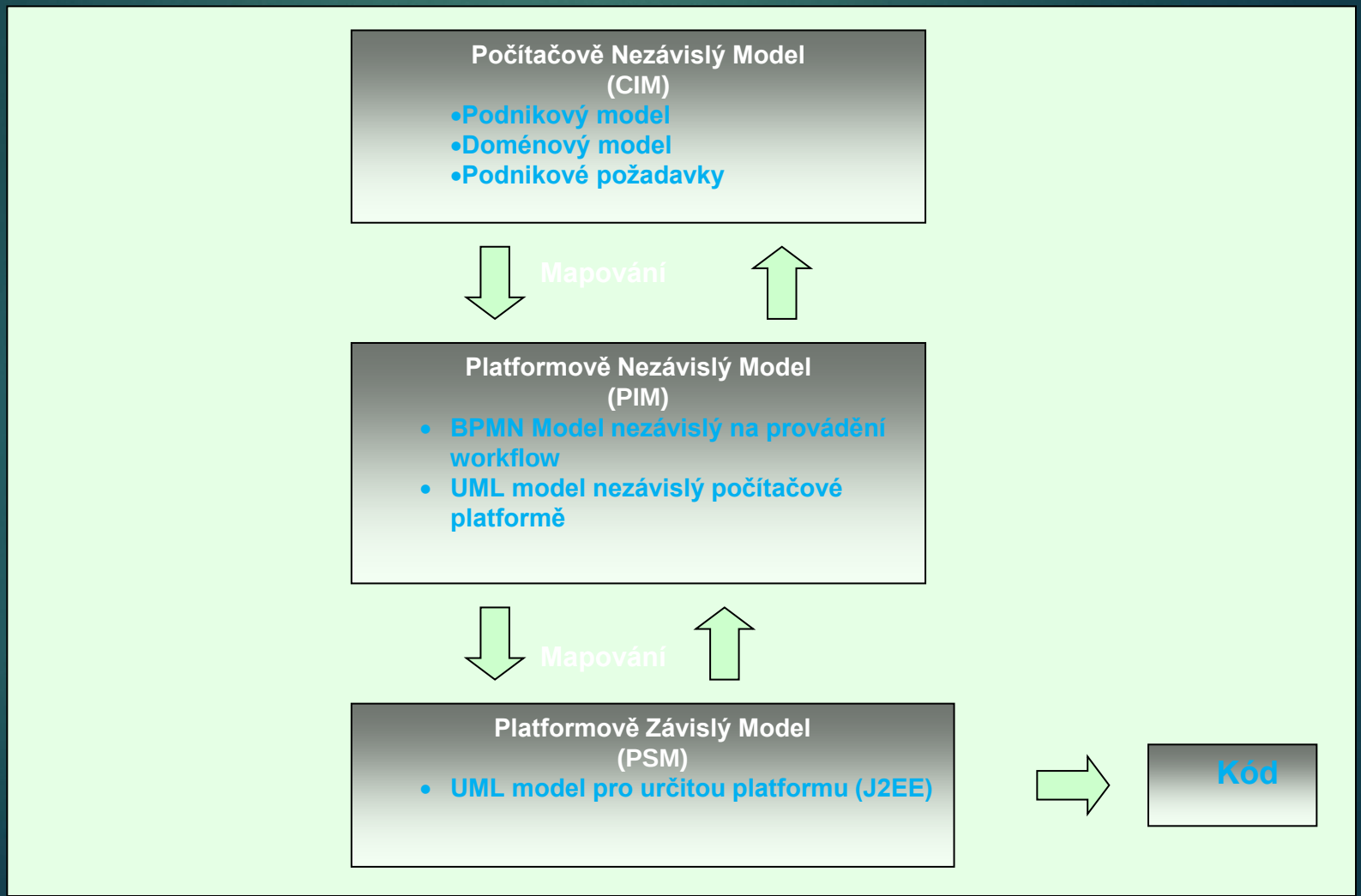
- Později se však princip soustředil na vrstvu **počítačově nezávislou, ve které jsou obsaženy podnikové aspekty společnosti**. Nové verze MDA nyní obsahují tři úrovně abstrakcí se vzájemnými vztahy podle následujícího obrázku.

Architektura založená na modelech (MDA)

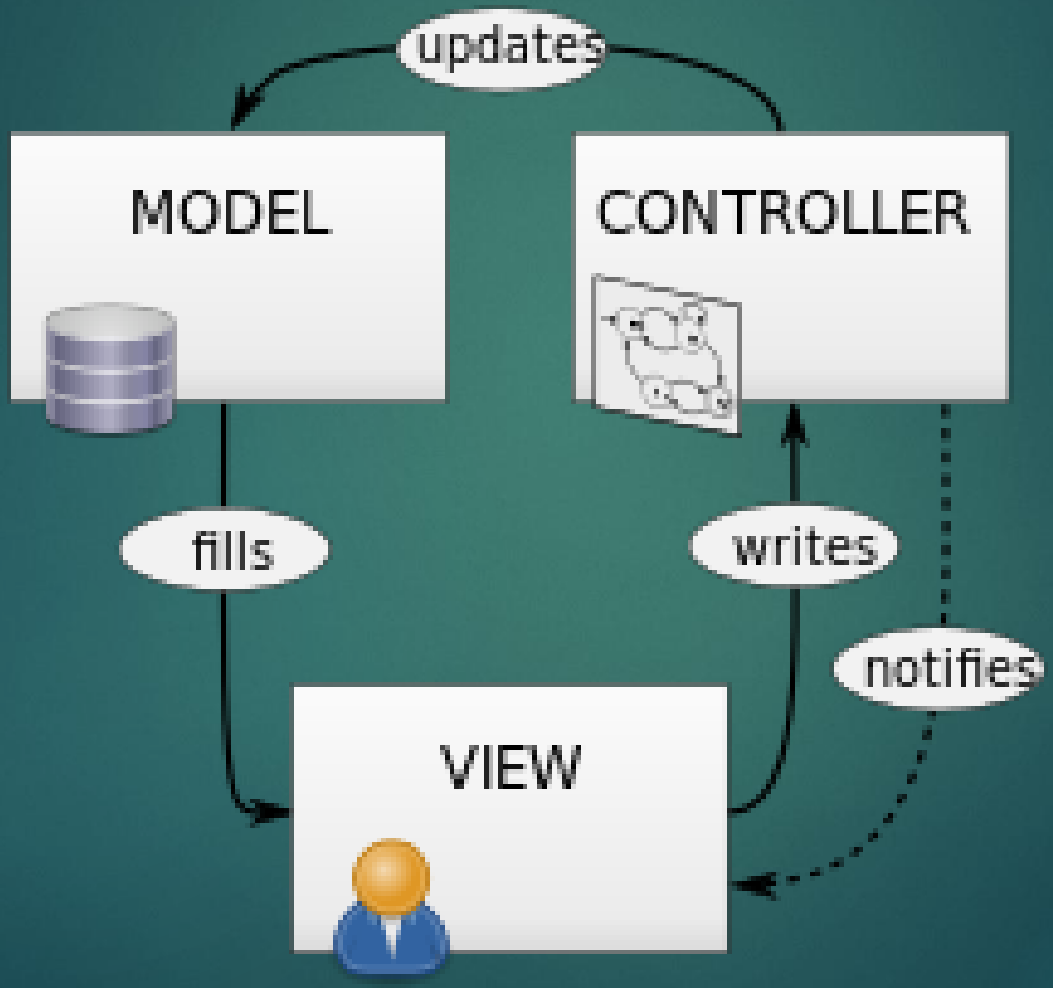
Pozn.:

- Aktuální problém všech modelovacích technik a nástrojů je míra preciznosti. Na jedné straně by měl být model abstraktní, lehce uchopitelný a srozumitelný. Na straně druhé stojí model, který obsahuje všechny podrobnosti, může být použit pro generování, ale už asi není tak přehledný a uchopitelný.
- Sdružení OMG řeší tento problém tak, že doporučuje vytváření modelů na různých úrovních –
  - ***doménový model*** (označen CIM – Computation Independent Model),
  - ***konceptuální model*** (PIM – Platform Independent Model),
  - ***logický model*** (PSM – Platform Specific Model) a nakonec
  - ***fyzický model*** – (kód).
- Na všech úrovních je třeba modely prezentovat pokud možno přesně, ale pouze v rozsahu potřebném pro tuto úroveň.

Architektura založená na modelech (MDA)



Architektura MVC



Architektura MVC

- MVC = Model-View-Controler je **architektonický vzor** pro webové aplikace
- Základní myšlenka: **oddělení logiky od výstupu**

- Celá aplikace je rozdělená na 3 typy komponent : modely, pohledy a kontrolory.

- **Model** obsahuje logiku (výpočty, dotazy, validace), model přijme parametry zvenku a vydá data ven. V objektově-relační databázi jsou to DB tabulky resp. třídy (mají atributy, metody)
- **Pohled** se stará o zobrazení výstupu uživateli (šablona obsahující HTML stránky a příkazy značkovacího jazyka s minimální logikou)
- **Kontroler** je prostředník, se kterým komunikuje uživatel, model i pohled, propojuje komponenty.

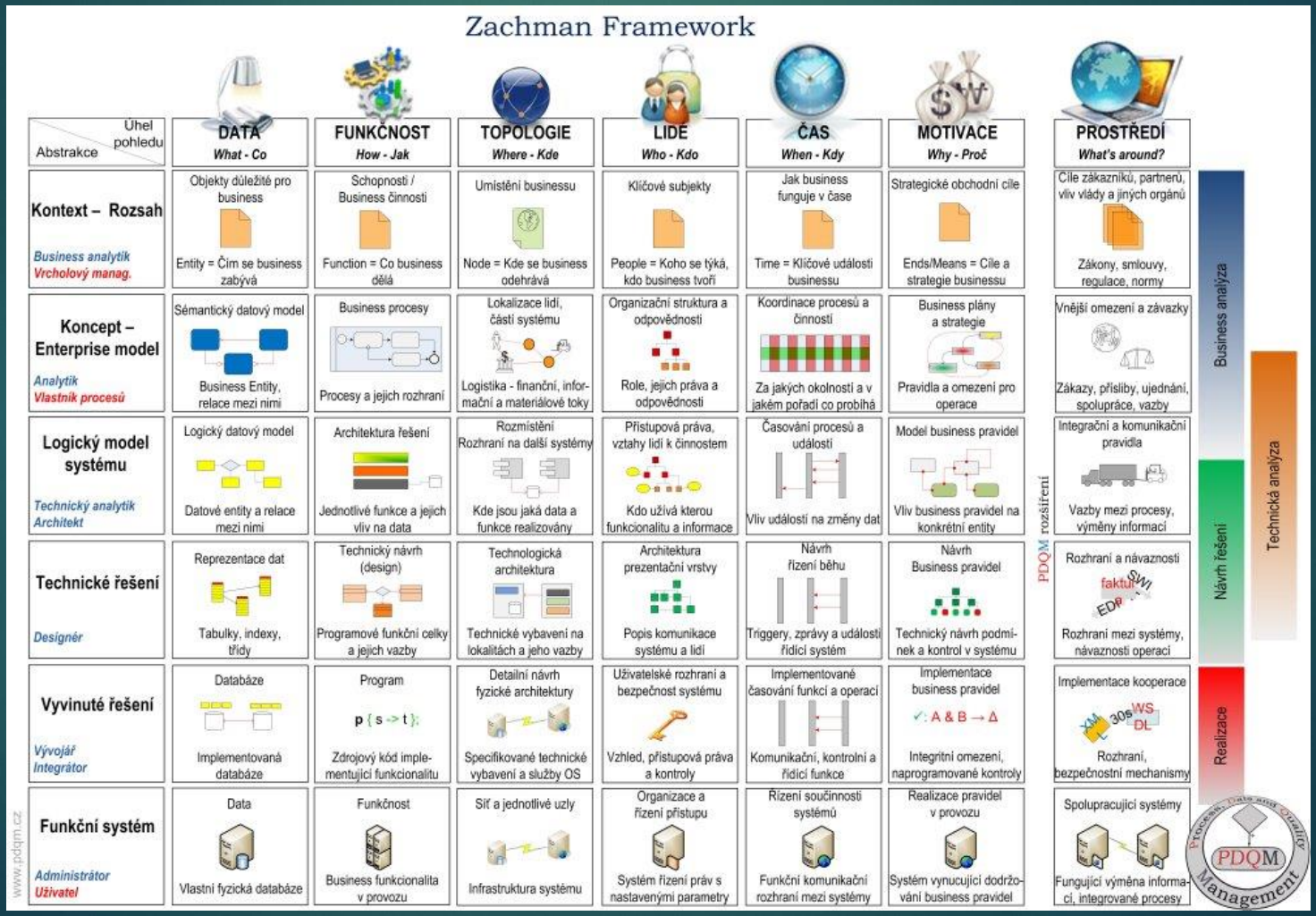
Architektura MVVM

- **Model-View-ViewModel** je **návrhový vzor (pro WPF Windows Presentation Foundation).** Nabízí řešení, jak oddělit logiku aplikace od uživatelského rozhraní. Kódu je pak méně, vše je přehlednější a případné změny nejsou implementační noční můrou. MVVM odděluje data, stav aplikace a uživatelské rozhraní.. Proto se v něm využívá binding a command – náhrada za uživatelské rozhraní řízené událostmi.
- **Model** popisuje data, se kterými aplikace pracuje. Pokud používáte Code-First Entity Framework, třídy, které se mapují na tabulky databáze, jsou Modely. Pokud referencujete webovou službu, třídy, které vám Visual Studio vygeneruje, jsou taktéž Modely. Model nesmí o stavu ovládacích prvků nic vědět.
- **View** reprezentuje uživatelské rozhraní v jazyce XAML. Může se jednat o okno aplikace, stránku, nebo ovládací prvek. Označuje se také jako UI, formulář, nebo prezentační vrstva ale jde pořád o to samé.
- **ViewModel** spojuje Model a View a drží si stav aplikace. Ovládací prvky jsou pomocí bindingu propojeny s ViewModelem a čerpají z něj svůj obsah. Provádí se v něm filtrování dat v závislosti na stavu aplikace. Aby se jeho vlastnosti propagovaly do View, musí implementovat rozhraní *INotifyPropertyChanged*. Jeho vlastnosti typu kolekce musí obdobně implementovat rozhraní *INotifyCollectionChanged*.

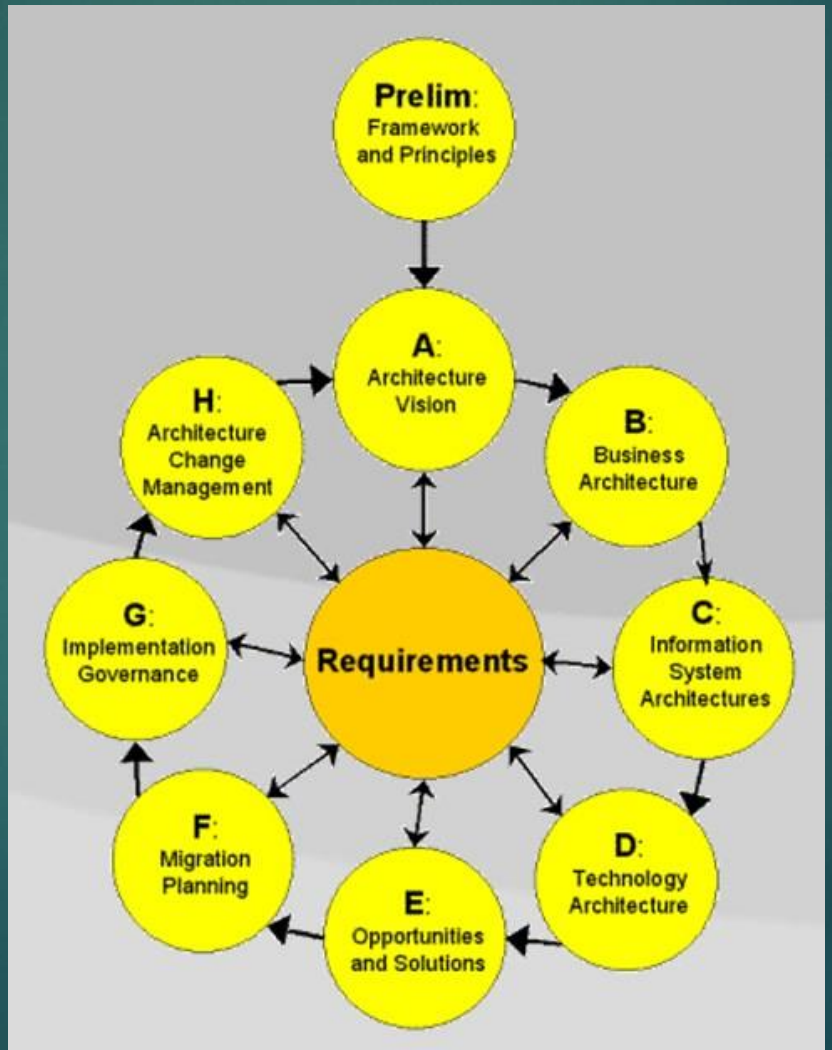
Architektura MVP (pro Android)

- **Model View Presenter** (MVP) je návrhový vzor, který vychází ze vzoru Model View Controller
- Hlavní rozdíl je v tom, co má na starosti mezivrstva – **Presenter/Controller**. Zatímco Controller definuje chování a typicky spravuje několik View, Presenter je ve většině případů v přímém vztahu pouze s jediným.
- Dalším rozdílem je, že View může v MVC komunikovat přímo s Modelem, ale v MVP je to Presenter, který má zodpovědnost za veškerou „komunikaci“ mezi View a Modelem.
  - **View** je zobrazovací vrstva, který uživateli ukazuje informace a veškeré input eventy jako zmáčknutí tlačítka, vložení textu, přesouvá k Presenteru.
  - **Presenter** má za úkol zpracovávat View input eventy a dotazovat se na data z Modelu. Když data dorazí do Presenteru, tak jsou zpracována a napojena na View.
  - **Model** je všechno, co si představíme ve spojení s daty – API, databáze, složky atd. Model má na starosti získávat zpět data a může být pověřen i sledováním změn v datech (jako v databázi), o kterých následně informuje Presenter.

Architektura podle Zachmana (podniková architektura – překonána novějšími modely např. TOGAF)



TOGAF - uznávaný framework pro podnikovou architekturu



Několik poznámek na závěr architektury IS

- **Co nevyřeší ani kvalitní IS?**
  - Špatné řízení společnosti
  - Špatné fungování společnosti
  - Chybný podnikatelský záměr
  - Mezilidské vztahy
  - Nedostatečné kvality zaměstnanců
  - Výrazné snížení nákladů
- **Co nám přinese kvalitní IS?**
  - Centralizace zpracování informací
  - Zjednodušení evidencí
  - Zvýšení efektivity práce
  - Mírné snížení nákladů
  - Rychlost získání informace
  - Vyšší objem a kvalita získaných informací

Volba správného IS (vlastnosti správného IS)

- je velmi odpovědná činnost a vůbec ne jednoduchá – obzvláště u ASW, se kterým nejsou žádné zkušenosti (ani dobré, ani špatné). Jaká tedy zvolit kritéria při výběru software? K základním lze zařadit:
  - **funkčnost** – tj. plnění požadovaných funkcí,
  - **provozovatelnost** (v různém prostředí, např. aplikace pro prostředí Windows XP nejsou provozovatelné v prostředí OS UNIX),
  - **spolehlivost** (ošetření chyb uživatele, žádné nedefinovatelné zastavení),
  - **ochranu dat**,
  - **otevřenost a modularitu** (pro perspektivní rozšíření funkcí),
  - **náročnost na obsluhu** (příjemné uživatelské prostředí),
  - **kvalitu servisních služeb,**
  - **přijatelnou dobu řešení úloh,**
  - **cenu.**

Vlastnosti správného IS (jiný zdroj)

Tyto atributy bychom měli požadovat od dobrého SW produktu:

- **Jednoduchost** (pro uživatele), **odolnost** (vůči chybám) a **stálost** (vzhledem k pracovním postupům a dokladům)
- **Udržovatelnost**
- **Provozní spolehlivost**
- **Výkonnost**
- **Použitelnost**
- **Respektování zákonů a zvyklostí**

- Děkuji za pozornost.