3. Data management, architektury systémů.

Osnova

1. Data management – principy, koncepce zpracování dat
2. Architektura klient – server
3. Třivrstvá architektura
4. Konfederativní systémy

Výklad

1. Data management – principy, koncepce zpracování dat

Kvalita dat a následně kvalita informací se s rozvojem Internetu a podporou manažerských informací stává stále významnější. Kvalita (jakost) je podle normy ISO 8402:

- souhrn znaků entity, které ovlivňují schopnost uspokojovat stanovené a předpokládané potřeby uživatele
- míra splnění požadavků zákazníka

Snažíme se najít takové atributy kvality (charakteristika dimense kvality), které:

- Mají význam pokud možno pro řadu různých aplikací pracujících s danými daty, jsou použitelné obecně (např. rozptyl).
- Jsou relevantní či zajímavé pro mnohé uživatele (ideálně všechny)
- Nejsou pokud možno přímo vázány na potřeby určité konkrétní aplikace (nejsou vstupem či parametrem algoritmů této aplikace).

Proč se problém kvality dat (a informací) stává rozhodující (až teď):

- Bez vyřešení problému, jak data ukládat, vyhledávat a prezentovat, nemělo dříve řešení otázky kvality dat smysl.
- Je nutné zajistit nejen ochranu dat, ale také zajistit jejich kvalitu a zavést procedury jak jednat, není-li kvalita dat ideální

V managementu se musí používat data, která nejsou zcela spolehlivá a relevantní. Podpora managementu se stává hlavním úkolem informatiky.

Míra

- Kvantitativní indikace rozsahu, množství, dimenze, kapacity, nebo velikosti nějakého atributu
- Počet chyb

Metrika

- Kvantitativní míra úrovně po kterou je dosažen atribut kvality
- Počet chyb nalezených / manday

Formáty metrik:

1. Příslušnost ke třídě (například výskyt určitého znaku, třeba čísla tramvaje)
2. Fuzzy (dobrý, lepší, nejlepší) – prvek uspořádané množiny, pro níž je jedinou přípustnou operací operace porovnání.
3. Intervalové (například teplota).

4. Číselné – jsou povoleny všechny aritmetické operace. Příkladem je rozsah souboru nebo jeho průměrná hodnota.

Metriky kvality dat jsou většinou fuzzy nebo číselné. Fuzzy metriky jsou subjektivní.

Řízení kvality dat (informací):

- Rozhodnutí o metrikách a procesech jejich měření (assessment) a nápravných opatřeních (control)
- Sběr a zlepšování jejich kvality (data cleaning)
- Odvozené procesy pro informace založené na zpracování daných dat
- Rozhodnutí o modernizaci nebo zrušení používaných metrik a postupů jejich měření

Objektivní metriky

- Objektivní metriky jsou metriky které lze vždy znovu vypočítat (po skončení vývoje) z dat, kterých se týkají.
- Jsou to často statistické charakteristiky datového souboru (rozsah souboru, průměr, rozptyl, výběrové momenty, např. Σixi3, korelace, atd.). Objektivní metriky jsou obvykle číselné.
- Objektivní metriky kvality dat odpovídají externím (explicitním) metrikám kvality softwaru ve smyslu ISO 9126-1 (např. délka programu)
- Často se používají při zlepšování kvality dat (čištění dat).
- Např: délka programů, počet metod, ...

Subjektivní metriky

- Subjektivní metriky jsou metriky hodnotící způsob, jakým data vznikla, případně kvalitu zdroje dat.
- Subjektivní jsou metriky hodnotící důvěryhodnost dat, stupeň jejich utajení, dostupnost, atd.
- Většinou jdou zjistit pouze během vývoje (in process metrics)
- Subjektivní metriky odpovídají metrikám interním (implicitní) podle ISO 9126
- Jsou de de facto subjektivním hodnocením vlastností dat experty založeným na zkušenostech a nikoliv na měření v běžném slova smyslu.
- Pro zkvalitnění dat je i v tomto případě nutno specifikovat, či standardizovat proces „měření“, mnohdy zákonem. Často s použitím komplikovaných dotazníků.
- Např. doba řešení, pracnost, průbeh velikostí týmu v čase, produktvita za jednotku času, počet selhání/výpadků za jednotku času, ...

Čištění dat

- Okrajová data (chyby měření) - ze souboru se vylučují data, která jsou zjevně nesprávná: úmyslně změněná, chybně zanesená (překlepy), nesprávného formátu.
- Chybějící data - do souboru se doplní chybějící data, aby bylo možno soubor rozumně zobrazovat (například časové řady) a přitom nedošlo k chybným výsledkům (k významným změnám charakteristik daného souboru).
- Vyloučení duplicitních dat – komplikované při nejednotnosti formátů (např. se díky tomuto problému dlouho nepodařilo vytvořit registr občanů) … díky duplicitním datům často instituce rozesílají duplikovaně korespondenci (zbyteční finanční ztráty)
- Sjednocení formátů
- A další …

Nejčastěji používané atributy kvality dat:

- Relevantnost (Relevance) – míra, do jaké míry data splňují účel, pro který jsou používána, týkají se daného problému.
- Přesnost (Accuracy) – jak přesná jsou používaná data (např. směrodatná odchylka). Kupodivu se neuvažují posunutá data
- Včasnost (Timeliness) – za jakou dobu lze data aktualizovat, jak jsou data aktuální.
- Dostupnost (Accessibility) – jak jsou již existující data dostupná. Obtížný problém díky nesmyslným předpisům pro ochranu privátních dat
- Porovnatelnost (Comparablity) – metrika hodnotící možnost porovnávat, ale také spojovat data z různých zdrojů.
- Koherence (Coherence) – metrika vyjadřuje, do jaké míry byla data vytvořena podle, z hlediska výsledku, kompatibilních pravidel
- Úplnost (Completeness) – metrika udávající jaká část potenciálních dat je zachycena v databázi, případně, zda výběr dat pokrývá „rovnoměrně“ celý výběrový prostor

Problémy s kvalitou dat při dolování dat a na webu – jak stanovovat míry kvality dat, jestliže:
- Daná míra má pro různé zdroje různé hodnoty
- Daná míra je i různě vyhodnocována (jiné procedury vyhodnocování)
- Daná míra se na některých zdrojích vůbec nevyhodnocuje
- Je pro nás lepší soubor s milionem údajů a rozptylem 2 nebo soubor s 100 údaji a rozptylem 1? Má smysl tyto soubory spojit?

Kvalita informací:
- někdy se ztotožňuje s kvalitou dat, ale není na to jednotný názor
- převažuje však názor, že jsou to dvě rozdílně oblasti, které spolu však úzce souvisí

Problémy s kvalitou dat při řízení:
- relevantnost a včasnost závisí na frekvenci zjišťování nebo jak je náročné data vytvořit
- kvalita dat může implikovat filosofii řešení (např. kritická cesta a krtický řetěz)

Problémy s kvalitou dat ve státní správě:
- duplicity
- ochrana dat
- legislativní ochrana kvality

Problémem anonimizace (aby nebylo možné zjistit osobu k níž data patří) je jak propojit k sobě patřící data pocházející z různých zdrojů a pořizovaných v různé době.

Brutální proces ochrany dat (BPOZ):
- data se de facto smí bez explicitního souhlasu používat a shromažďovat jen k účelům, pro která byla pořízena a to jen pověřenými institucemi
- toto podstatně nezlepšuje kvalitu dat, existují kanály, kterými budou informace stále unikat a nikdo tomu nezabrání (obchodní rejstříky, katastr, černý trh, mobilní telefony, sociální sítě, …)
- naopak nás zbytečně ohrožují
    - zdravotnictví a sdílení údajů mezi lékaři atd. (viz. příklad s pervitinem a chybné medikace)
    - evaluace vzdělávání

2. Architektura klient – server (http://cs.wikipedia.org/wiki/Klient-server)

Vsuvka o architektuře: je to struktura systému (komponenty, jejich vlastnosti, spolupráce, rozhraní navenek). Architektura ovlivňuje uživatelské vlastnosti systému, dostupné operace, techniky specifikací a metodologii vývoje. Většinou prostředek dekompozice, porozumění, analýzy (např. konzistentnost a kvalita návrhu). Důležité pro: zvládnutí vývoje, znovupoužitelnost částí, údržbu, podporu decentralizace, usnadnění řízeení (procesy, malé etay, inkrementální vývoj, minimální rozsah, rozšiřování).

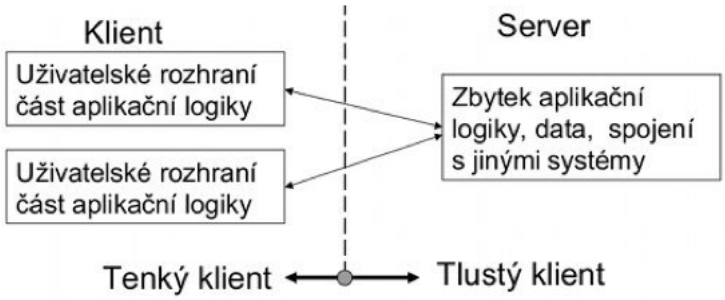
Patří do kapitoly dekompozice IS do sítě spolupracujících aplikací, které spolu navzájem komunikují. Jde o síťovou architekturu oddělující klienta od serveru. Komunikace mezi nimi probíhá po síti. Příkladem tohoto modelu je například web. Máme webový prohlížeč (klient) a webový server (server), kterého „žádáme“ o informace a komunikace probíhá po sítí.

Charakteristika klienta:

- Aktivní; Posílá žádosti serveru; Čeká a dostává odpovědi; Obvykle je připojen k malému počtu serverů najednou; Obvykle komunikuje přímo s koncovými uživateli, pomocí grafického uživatelského rozhraní

Charakteristika serveru

- Pasivní; Naslouchá na síti a reaguje na žádosti připojených, autorizovaných klientů; Při přijetí požadavku jej obslouží; Může vzdáleně instalovat/odinstalovat aplikace a přenášet data ke klientům



Alternativou k architektuře klient-server je peer-to-peer (P2P). U této architektury může každý hostitel nebo instance programu fungovat zároveň jako klient i jako server (mají rovnocenné postavení i zodpovědnost).

Výhody:

- Ve většině případů architektura klient-server rozdělí jednotlivé úkoly a zodpovědnosti počítačového systému mezi několik počítačů které spolu komunikují pouze prostřednictvím sítě.
- Tím vzniká další výhoda této sítě, a to snadnější údržba. Například je možné nahradit, opravit, modernizovat, přemístit server, aniž by to klienti poznali, nebo tím byli nějak ovlivněni. Tato nezávislost na klientech se nazývá zapouzdření.

- Všechny údaje jsou uloženy na serverech, které jsou mnohem bezpečnější než většina klientů. Servery mohou lépe kontrolovat přístup a zdroje, to zaručuje, že přistupovat a měnit data mohou pouze oprávnění klienti.
- Vzhledem k tomu, že se data ukládají centralizovaně, aktualizování údajů je mnohem jednodušší než u P2P sítí. V P2P sítích je potřeba updatovat data na každé stanici zvlášť, což je pomalé a způsobuje množství chyb, protože mohou mít tisíce nebo miliony klientů.
- → **úspora nákladů na HW klientů, snazší správa (upgrade, opravy, ...), bezpečnější**

Nevýhody:

- Velkým problémem je přetěžování sítě. Vzhledem k tomu, že počet souběžných požadavků klientů na daný server se zvyšují, server se může snadno přetížit. Naproti tomu u P2P sítí se šířka pásma zvětšuje s množstvím klientů, protože každý klient tvoří uzel sítě.
- Architektura klient-server není tak robustní jako sítě P2P. Pokud dojde k výpadku serveru, žádosti klientů nemohou být splněny. V P2P sítích jsou zdroje obvykle distribuovány mezi více uzlů. Dokonce i když více uzlů přeruší sdílení dat, mělo by být možné stáhnou data od zbývajících uzlů.
- Neefektivní při malém počtu klientů.
- → **výpadek ovlivní všechny, cena přechodu, přetížení serveru, může být neefektivní, některé služby mohou být lépe proveditelné lokálně**

3. Třívrstvá archikektura

Jde o typ architektury klient-server, používaný často např. u webových aplikací. Obecně se používá u aplikací pracujících s daty (s velkým množstvím dat). Výhodou architektury je, že odděluje jednotlivé vrstvy tak, aby na sobě nebyly závislé.

Prezentační vrstva

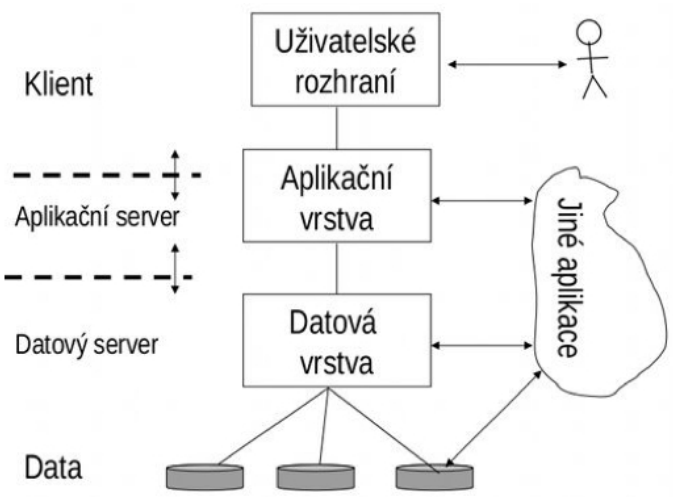
- Zobrazuje informace pro uživatele, většinou formou grafického uživatelského rozhraní, může kontrolovat zadávané vstupy, neobsahuje však zpracování dat.

Aplikační vrstva (též Business Logic)

- Zde leží jádro aplikace, její logika a funkce, výpočty a zpracování dat.

Datová vrstva

- Tuto vrstvu tvoří nejčastěji databáze, která data uchovává, zpřístupňuje a zaručuje jejich konzistenci. Může zde být ale také (síťový) souborový systém, webová služba nebo jiná aplikace.



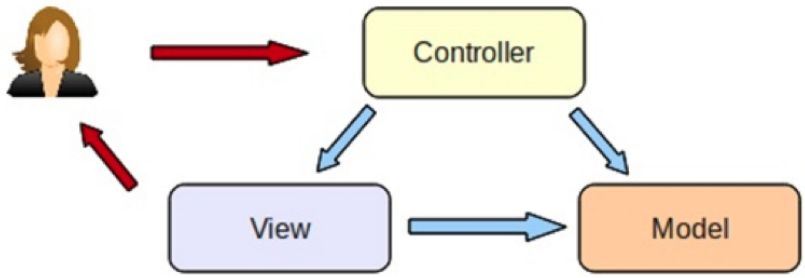
Přínos architektury klient/server a třívrstvé architektury

- pružnější rozdělení práce
- lze použít horizontální (více serverů) i vertikální (výkonnější server) škálování
- aplikace mohou běžet na levnějších zařízeních
- na klientských stanicích lze používat oblíbený prezentační software
- standardizovaný přístup umožňuje zpřístupnit další zdroje
- centralizace dat podporuje účinnější ochranu
- u třívrstvé architektury centralizace údržby aplikace, možnost využití sdílených objektů (business objects) několika aplikacemi

Architektura MVC

Model-view-controller je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace

některé z nich má jen minimální vliv na ostatní. Vychází z tří-vrstvé architektury.



Existuje přímá vazba mezi Controllerem a Modelem (Controller upravuje data Modelu). Dále mezi View a Modelem (View zobrazuje data Modelu). Často bývá i vazba mezi Controllerem a View v tom smyslu, že Controller informuje View o změnách. Nikdy by ale neměla být vazba z Modelu na ostatní dvě komponenty.

Model

- tvoří datovou vrstvu a business logiku (doménovou logikou). Neřeší formu ukládaných dat (relační DB, objektová DB, XML souborech apod.). Business logika představuje funkce pro manipulaci s uloženými daty, základní operace, výpočty, business pravidla, validační kritéria atd.

View

- představuje uživatelské rozhraní, pomocí kterého uživatel interaguje se systémem. Zobrazuje data požadovaným způsobem. Lze definovat různé pohledy na stejná data (např. tabulka nebo graf), různé barevné profily, profily s ohledem na výstupní zařízení (malá obrazovka PDA vs. velká obrazovka monitoru) apod.

Controller

- obsahuje kód, který řídí procesy uvnitř aplikace. Stará se o realizaci funkčních procesů a o správné provázání ostatních komponent aplikace do jednoho funkčního celku. Příklad: Pokud dojde k zadání nových dat ve View, musí se postarat o jejich zpracování Modelem a následnou aktualizaci (překreslení) těch View, které jsou na nových datech závislé.

Příklad interakce uživatele s MVC systémem:

1. Uživatel vykoná akci v uživatelském rozhraní.
2. Akce je zachycena Controllerem, který promítne změny do modelu. Controller nebo Model informuje View.
3. View aktualizuje uživatelské rozhraní a promítne tak provedené změny zpět uživateli.

4. Servisně orientovaná architektura (http://cs.wikipedia.org/wiki/Service_Oriented_Architecture, http://kore.fi.muni.cz/wiki/index.php/SOA), Konfederativní systémy

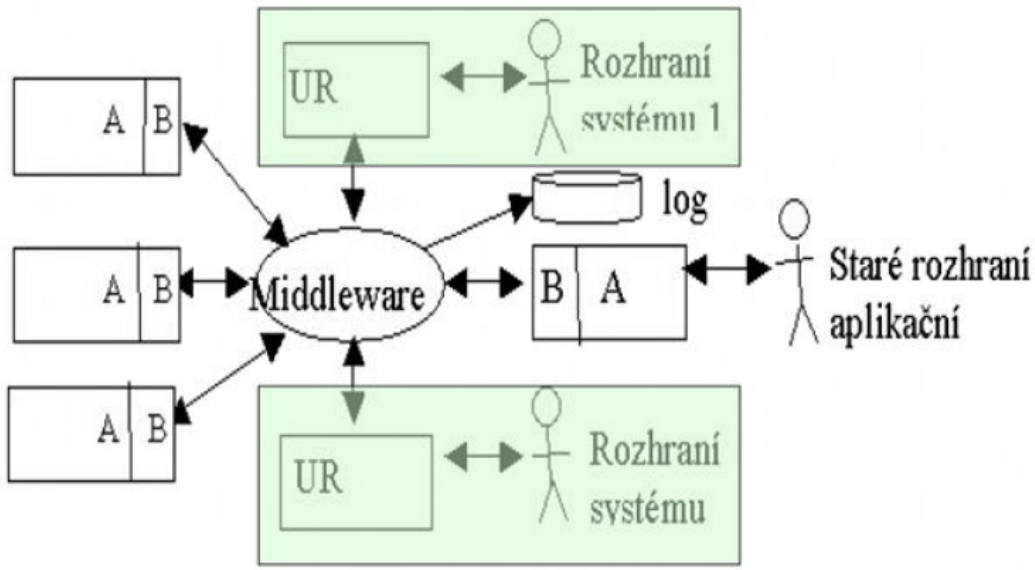
SOA je je sada principů a metodologií, která doporučuje skládat složité aplikace a jiné systémy ze skupiny na sobě nezávislých komponent poskytujících služby → máme tedy samostatné služby a vzájemně je propojujeme. SOA je podobná s OOP, ale principy jsou realizovány v jiné vrstvě architekruty.

(služba = autonomně pracující komponenta)

Motivace: dříve byl systém na jedné platformě a stačilo komunikovat pomocí API jedné technologie. Dnes je situace odlišná, systémy musí být schopné komunikovat napříč technologiemi a systémy.

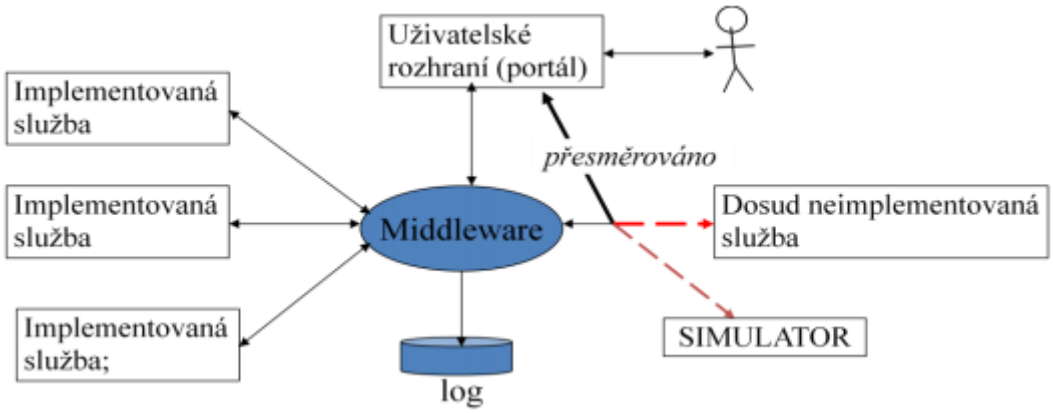
Charakteristika:

- Klíčové paradigma současného softwaru
- Komponenty spolupracují předáváním zpráv
- Základní styl komunikace je asynchronní, sekvenční je nadstavba
- Virtuální síť p2p velkých volně spolupracujících autonomních komponent, spolupráce podobná spolupráci služeb v reálném světě,
- Služby jsou většinou stále k dispozici, asynchronně reagují na požadavky z různých zdrojů, jsou používány jako černé skříňky



A – aplikační služba,
B – její rozhraní (primární brána)
UR je uživatelské rozhraní, např. portál

Prototypování v SOA:



Zprávy lze přesměrovat beze změny implementovaných služeb buď na uživatelské rozhraní (efektivní prototyp, použitelné i za běhu systému), nebo dokonce na simulátor (ladění RT systémů).

Konfederativní systémy

- ALIANCE - Na začátku komunikace se partner musí vyhledat. Typické pro e-komerci, reservační systémy atp. Je nutné pužívat (celosvětové) standardy
- KONFEDERACE - Partneři jsou zpravidla známi. Široká paleta aplikací, historicky nejstarší SOA systémy (soft RT systémy, e-government, řízení globálních společností, výrobní systémy, koalice podniků a zdrav. zařízení). Protokoly komunikace mohou být proprietární. Komunikace může být i jiného typu, než výměna zpráv založená na internetu. Klíčovou výhodou konfederací je, že rozhraní služeb může být uživatelsky orientováno. Konfederace mohou být volné a úzce vázané, otevřené a uzavřené.
- UNIE - Konfederace ve které musí být rozhraní služeb uživatelsky orientováno

Existují i meistupně mezi A a K. Často je jádro konfederace a periferie aliance.

Proč použít konfederaci:

- Chceme-li použít uživatelsky orientované nebo existující rozhraní a middleware jiný než webovský
- Propojování systémů různých zdrojů
  - existující aplikace, vývoj, třetí strany
- a různých vlastníků
  - organizační subsystémy, decentralizace
  - spolupráce samostatných podniků
- a různých technologií a úkolů
- Příliš velký systém na monolit
- Chceme inteligentní rozhraní, a někdy i dávkové techniky

Příklady konfederativních SOA (od nejvolnějších vazeb k nejtěsnějším):

- e-komerce
- IS státní správy
- spolupráce zdravotnických zařízení
- IS globálního podniku
- lokální divize globálního podniku, menší podnik

- řízení procesů (soft real-time)
- OO aplikace

Kdy použít SOA:
- Jako prostředek podpory distribuovanosti
- Jako prostředek dekompozice na daném místě, v daném počítači
- Jako prostředek sdružování kapacit (gridovské systémy, viz projekt CETI)
- Podpora inkrementálního vývoje a jiné SW inženýrské výhody, např. znovupoužitelnost, kombinace produktů různého původu
- Podpora otevřenosti, CRM a SCM
- Aby to vůbec šlo napsat (Neexistence SOA v sedmdesátých létech byla jedním z důvodů krachu projektu protiraketové obrany SALT). Velký systém musí být budován po částech.
- Spolupráce existujících systémů které je lepší použít tak jak jsou (nelze je rychle přepsat, politika, autonomní systémy)

Výhody SOA:
- Chyby zůstanou lokalizované, modernizaci lze provádět po částech. Stávající systémy mohou sloužit nadále bez podstatnějších změn, lze integrovat produkty třetích stran
- Flexibilita. Produkt se dá snadněji upravit pro trošku jiné účely
- Škálovatelnost. Lze snadno doplňovat nové služby a také je klonovat
- Autonomie částí zvětšuje odolnost systému proti výpadkům. Dosavadní uživatelé existujících služeb jim mohou věřit, protože se v podstatě neměnní. Nemusí se učit příliš nového a mohou důvěřovat i datům.
- Úspory nákladů z důvodů znovupoužitelnosti, použití produktů třetích stran a snadnějšího vývoje všeobecně
- → **chyby zůstanou lokalizované, flexibilita, znovupoužitelnost částí, škálovatelnost, autonomní vývoj částí, úspory nákladů, inkrementální vývoj, modifikovatelnost a udržovatelnost**

Avšak proces zavádění SOA je třeba důkladně sledovat a řídit - kontrolovat a monitorovat služby, jejich výkon, spolehlivost a zejména bezpečnost. Jinak může dojít k neefektivnímu řešení, které věci komplikuje místo, aby je usnadňovalo. Také může být pro někoho obtížné příjetí filosofie SOA.

Závěr

- knížka *Informační Systémy* (Jaroslav Král)
- vypracováné otázky TIS I a TIS II