Projektování informačních systémů 6

Implementace projektu

Doc. Mgr. Petr Suchánek, Ph.D.

Doc. RNDr. Ing. Roman Šperka, Ph.D.

Převzato od: Ing. Dominik Vymětal, DrSc.

Zpracování cílového konceptu řešení

- Cíl:
  - Návrh architektury systému
  - Obsahuje
    - podrobný popis funkcí systému (např. příklad procesní model)
    - Architekturu datové základny
    - Návrh HW a infrastruktury
    - Návrh konverze dat
    - Aktualizaci spotřeby času a zdrojů
    - Údaje o množství (počty uživatelů, počty dokladů, charakteristika zákazníků, zboží , skladů, dodávek , hospodářských středisek,
    - Bezpečnost systému, systémy oprávnění
- Výstup:
  - Schválení architektury, časových plánů , plánu zdrojů, aktualizace harmonogramů a ceny.

Příklady Konica Minolta

Přechod k realizaci

- Bez ohledu na typ metodologie se po zpracování návrhu nového řešení a následné alokaci zdrojů přechází k realizaci IS

Příprava nového řešení (Realizace)

- Vychází z návrhu řešení vzor sales
- Obsahuje
  - Návrh a instalaci HW
  - Instalaci a konfiguraci standardních modulů SW
  - Programování, instalaci, úpravu zákaznických modulů / programů
  - Iterativně
    - Vytvoření prototypů databází, obrazovek
    - Prověření prototypů
  - Iterativně
    - Vytvoření prototypů funkcí
    - Prověření prototypů funkcí
  - Vytvoření základní dokumentace
- Zpravidla přechází do etapy akceptace

Architektury IS

- Praktická příprava provedení vychází také z celkové architektury systému
- Různé softwarové balíky mají různou architekturu
- Client – Server – v současné době základní
- SOA –opakované volání služeb
- Agenti – nové paradigma

C/S Něco z historie

- IBM System Application Architecture – metodika připojení terminálů (PC) k mainframe strojům
- Původní architektura Mainframe není S/C, uživatel pracuje ze stanice pevně připojené k hlavnímu počítači a zasílá jednotlivé znaky nebo jejich sady přímo.
- S rozvojem GUI se tato architektura přežila, ještě nyní doznívá u emulací např. na IBM AS400
- Na počátku 80. let jako označení připojení první PC do sítě
- V závěru 80. let se C/S začala prosazovat
- Mezistav: Sdílení souborů na serverech
- V průběhu 90. let v souvislosti s OO přístupem k tvorbě IS byla všeobecně přijata, a to i na velkých počítačích

C/S Definice

- Obecná:
  - Client / Server je architektura založená na tom, že klient žádá servis od serveru.
- Podrobnější:
  - Je to síťová architektura, která odděluje procesy serveru a klienta. Každá instance klienta může na server zasílat požadavky.
  - Existují specifické typy serverů jako: webovské, aplikační, file servery, terminal servery, mile servery.
  - Moderní software prakticky vždy umožňuje provoz dle architektury C/S
- Další aspekty:
  - C/S obvykle znamená rozdíl mezi procesy a stroji. Zpravidla je to tak, že server a klient jsou rozdílné počítače
  - Na druhé straně je v této architektuře dáván důraz na procesy více než na hardware.
  - Některé typické C/S aplikace: e-mail, FTP, prohlížeče

C/S definice II

- Charakteristiky Serverů:
  - Jsou „pasivní“ – (Slave)
  - Čekají na požadavky
  - Po obdržení požadavku jej zpracují a zašlou odpověď
- Charakteristiky Klientů:
  - Jsou „aktivní“ – (Master)
  - Zasílají požadavky
  - Čekají na to až Server na požadavek odpoví
- Peer to Peer Architektura
  - Každý uzel nebo běžící program je současně Serverem i Klientem a oba mají stejné úkoly.
- Vztah k metodikám modelování systému
  - Tato architektura může mít problém s metodikou ERA, lépe se popisuje OO metodikou, vyžaduje specifické přístupy

Klient a jeho typy

- Klient je počítač, který dostává služby od jiného počítače pře síť
- Klient je i nadále široce využíván na internetu, u řady aplikací dnes dochází k využití prohlížečů jako standardních klientů
- Typy klientů:
  - „tlustý“ (fat) klient – provádí zpracování převážné části dat sám
  - „tenký“ (thin) minimalizovaný klient. Má většinou za úkol provádět pouze prezentaci dat dodaných aplikačním serverem, který zajišťuje většinu zpracování.
  - Hybridní, např. provádí zpracování dat, ale vlastní data jsou na serveru.

Tenký klient jako typ aplikace

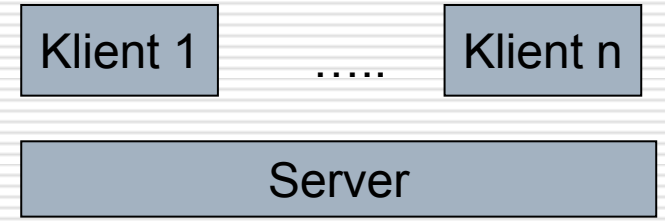
- Volba typu klienta je důležitá pro návrh celé aplikace
- Tenký klient jako aplikace komunikuje s aplikačním serverem a dostává převážnou část logiky z aplikačního software běžícího na serveru někde v síti.
- Jiné dělení: kde vlastně běží aplikace – na HW klienta nebo na HW aplikačního serveru
- Příklady:
  - MS Office na serveru a jejich instance na klientech
  - Virtuální image OS na serveru, data na serveru, vlastní výpočty na mobilních klientech přes VPN
  - Přístup přes WEB k SAP nebo Navision

Srovnání typu klientů

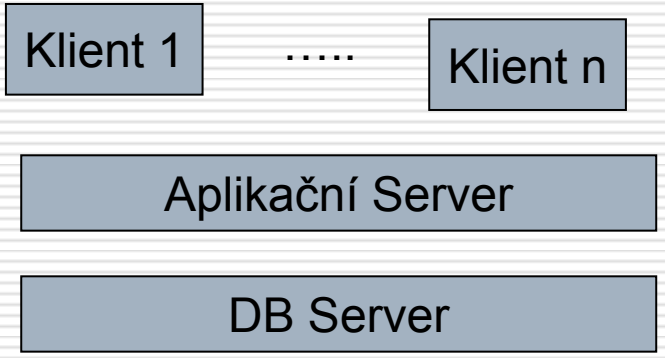
- Výhody tenkých klientů
  - Nižší náklady na administraci IT, jednodušší zabezpečení
  - Nižší náklady na hardware
  - Nižší spotřeba energie
  - Lepší využití zdrojů
  - Menší nároky na šířku pásma sítě
  - Nepoužitelné pro zloděje
  - Jednodušší upgrade a změny software
- Výhody tlustých klientů
  - Menší nároky na servery, úspora nákladů na pořízení
  - Menší nároky na síť v případě multimediálních aplikací
  - Vyšší pružnost, v případě Windows značně složité

Vrstva v architektuře Client/Server

- Dvě vrstvy (two-tier)



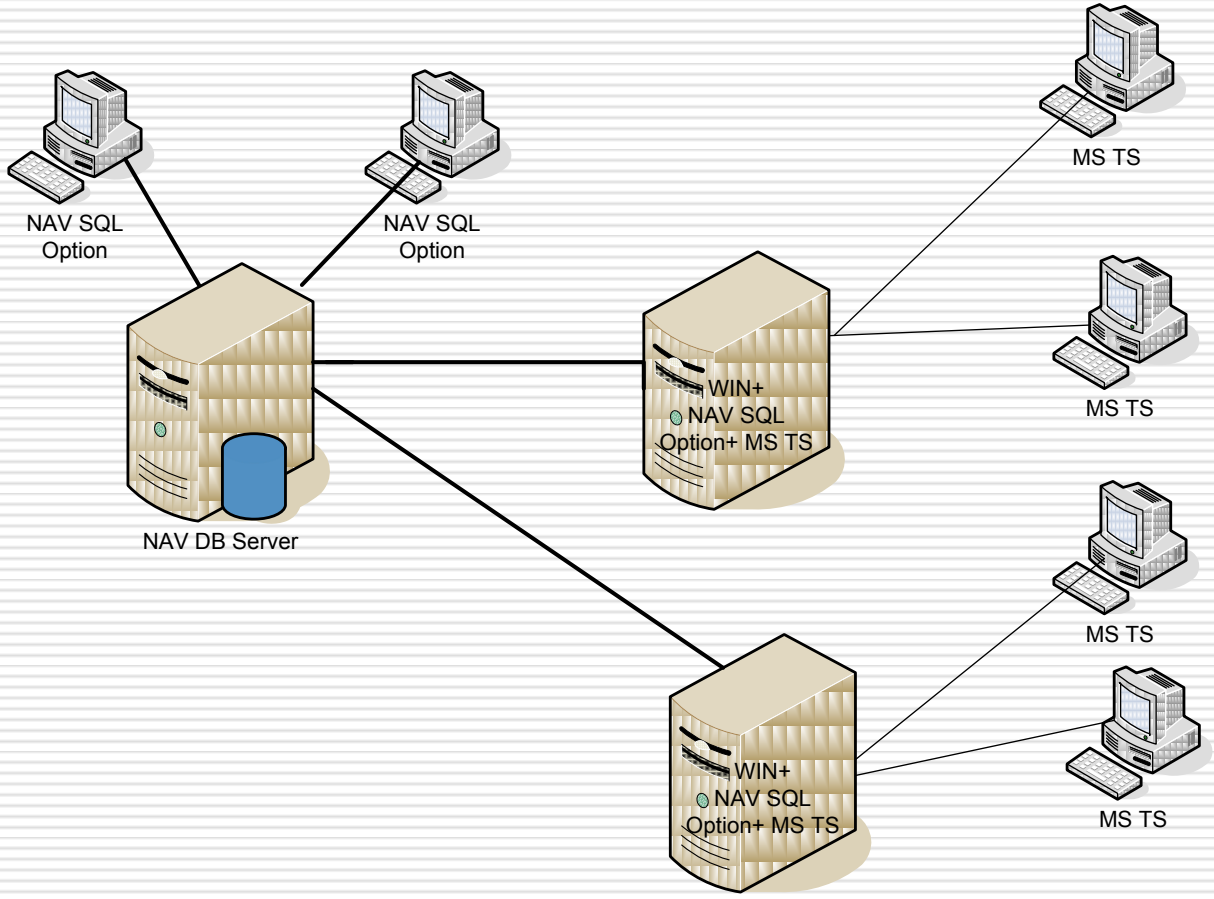
- Tři vrstvy (Three-tier)



Třívrstvá architektura - důsledky

- Klient zajišťuje prezentaci aplikační logiky a dat
- Aplikační server je oddělen od uživatele i od databáze
- Aktivity přináležející určitému uživateli mohou být plně „privatizovány“ – důsledky v systému oprávnění
- Možnost „předpřípravy“ dat (application pre-processing)
- Celá logika aplikace může být oddělena a spravována z jednoho místa (údržba a nové požadavky)
- Databázové servery mohou používat nezávislý software
- Může vzniknout z dvouvrstvé architektury

C/S architektura v případě MS Dynamics NAV



Důsledky C/S architektury

- Procesní orientace
  - Hierarchická struktura funkcí nevyhovuje
  - Dialog je řízen člověkem
  - Práce je vedena stylem objekt - akce
  - OO přístupy se hodí lépe
- Model uživatelských objektů
  - UO diagram tak jak je chápe uživatel
  - Uživatelské rozhraní: okna a akce nad nimi (tomu odpovídají akce nad databází)
  - Diagram pohybu mezi okny
- 3 typy modelů
  - Procesní (diagram datových toků)
  - Datový (entity a vazby)
  - Entity a události (diagram životního cyklu entit)

Programátorské důsledky C/S architektury

- Vysoká závislost na síti a její výkonnosti
  - LAN – kabeláž min kat5 100Mbit/s (mezi servery min 1Gbit)
  - WAN – min 2Mbit
- Nutná změna z procedurálního modelu na model řízený událostmi
- Změna způsobu přípravy uživatelského prostředí (na prvky jsou navěšeny triggery, lišty, ikony …)
- Programátor :
  - Namaluje obrazovky
  - Nastaví triggery a jejich atributy
  - Zorganizuje externí i interní podprogramy
  - Musí uvažovat s vícenásobnou použitelností (zejména webovské přístupy)

Problematika rozdělení na část klient a část server

- Kontrola na relačnost a konzistenci datového modelu
  - Mohlo dojít k porušení během specifikace UO
- Specifikace ochrany dat a přístupů
  - V důsledku vícenásobné použitelností může dojít k problémům
  - Příklady moduly HR a provizní systém, ceny pro přímý prodej a dealery atd.
- Problém více databázových strojů
  - Replikace, synchronizace,
  - Zvážení účelných duplicit a důsledků v reportingu
- Rychlost kritických aplikací
  - Rychlost DBMS
  - Rychlost infrastruktury
  - Způsob práce s okny (zdroje koncových strojů)
  - Kritické uživatelské úkoly (filtry, počítaná pole, hledání, třídění)

Některé nevýhody C/S

- Systém management (konzistence a aktuálnost ve velké síti)
- Závislost na síti – nebezpečí nákladů na duplicitní zařízení
- Nákladné financování spojení aplikačních serverů s klienty (MS Terminal Server, Citrix)
- Optimalizace a doladění DB serverů není jednoduché
- Současná tendence u aplikačních C/S architektur vede k webovským přístupům (cloudy)
  - Bezpečnost, mobilita, oprávnění, demilitarizované zóny, otázky uvolněných portů, …

Zpět k vlastnímu průběhu projektu

- Zopakujme si
  - Model vodopád
  - Model spirála
  - Prototypování
  - Agilní metodiky
- Velké projekty stále využívají mix vodopádu, spirály a prototypování

Použití různých metod v různých etapách projektu

Design stage	Process modeling	Value chain modeling	REA including dynamization	Modeling or simulation by agents	Spiral design	Prototypes	Agile methods
Strategic goals definition							
Pre-feasibility study							
Feasibility study							
Project targets setup							
Analysis and design							
Detail specification							
Customizing and programming							
Testing and user education							

Prototypování

- Princip:
  - víceúrovňové prototypy, například prototypy GUI a poté prototypy hlavních funkcí reagujících na události v GUI
  - opakované diskuze s uživateli a úpravy prototypů
- Výhody:
  - včasné zapojení uživatelů do přípravy systému a získání jejich pozitivní motivace
  - možnost dolaďovat systém tím, že se nejdříve testují hlavní funkce a podle připomínek uživatelů se jejich detailní funkce přizpůsobují uživatelským potřebám
  - vedení projektu má větší přehled o postupu a plnění jednotlivých pracovních kroků
  - uživatel dostává to co uviděl
  - možný rozvoj řešení po etapách a iteracích

Prototypování

- **Nevýhody**
  - nutnost kompetentního vedení projektu (nebezpečí zacyklení a ztráty času)
  - při nedostatečném naplánování může dojít k neorganizovanému vývoji, kdy se požadavky uživatelů zavádějí bez vazeb na jiné části systému nebo se porušují již odzkoušené funkce
  - tím mohou stejně jako později v etapě testování vznikat sekundární chyby
  - uživatel může mylně dojít k závěru, že systém je již téměř hotov a podceňuje fázi testování
  - často dochází k tomu, že je nedoceněna práce na dokumentaci
  - je zde určité riziko, že uživatelé své připomínky k prototypům mohou považovat za chyby sytému
  - tím může vzniknout negativní motivace

Agilní metodiky

- Hlavní principy:
  - Iterativní vývoj IS. Těchto iterací může být provedeno větší množství v co nejkratším čase.
  - Předem připravené testy. Při každé iteraci je produkt podroben testování. Příprava testů musí být velice pečlivá, aby se zabránilo sekundárním chybám v již otestované předchozí verzi.
  - Co nejširší komunikace vývojářů a budoucích uživatelů.

Některé agilní metodiky

- Rodina Unified metodik: Unified process (též USDP), Rational Unified process (IBM) iterace, OpenUP
  - Inception – elaboration – construction - transition
- Lean software development process:
  - „Vše co není k užitku zákazníkovi je odpad“
  - Co nejvíce testů – rozhodnutí co nejpozději – dodej co nejrychleji – podporuj tým – zabuduj integritu – pohlížej na celek
- SCRUM - mlýn: malé přehledné části, které se realizují ve „sprintech“
- Extrémní programování – dotažení známých metod „do extrému“

Extrémní programování

- Aktivní účast zástupce uživatele. Požadavky uživatele se shromažďují do „User stories“.
- Iterativnost postupu.
- Neustálé využívání zpětné vazby, maximální komunikace v projektovém týmu, neustálé testování jednotlivých modulů a vlivu změn na celkový výsledek, komunikace s uživatelem.
- Maximální jednoduchost, programuje se jen to co je v daný moment potřebné, neustále se zkoumá, zda by mohla existovat ještě jednodušší varianta řešení.
- Začíná se programovat od jednotkových testů.

Extrémní programování

- Stejný modul programují dva programátoři, kteří sedí u jednoho počítače. Diskuzí dvojice nad tvořeným programem se vyloučí chyby a dosáhne zjednodušení kódů.
- Krátké iterace – provede se jedna změna v programu a ihned se ověřují výsledky.
- Zdrojové kódy jsou vlastnictvím všech zúčastněných programátorů, všichni jsou společně zodpovědní za výsledek.
- Neprovádí se žádná optimalizace, jestli má optimalizace proběhnout, pak se provede na konec.
- Integrace napsaných kódů do existujícího řešení provádí v jeden okamžik pouze jedna dvojice programátorů – tím se vyloučí diskuze, ze kterého kódu vznikla případná chyba. Integrace se provádí co nejčastěji.
- Na počátku se nepíše zdrojový kód řešení, ale připravují se jednotkové testy modulů.
- Jednotliví programátoři ve dvojicích se obměňují.

Aspektové přístupy

- Od poloviny 90.let
- V zásadě využívají myšlenky objektového programování, bývají však označovány za „post objektové“ přístupy, což je dáno dobou jejich vzniku.
- Vznik aspektového programování byl vyvolán potřebou modularizovat často se opakující části programového kódu, které se nacházejí v různých místech aplikací.
- Tyto části se označují za „záležitosti“- anglicky concerns a jejich „rozptýlený“ výskyt v aplikacích vede k jejich označení „crosscutting concerns“
- Oddělení jednotlivých „záležitostí“ do programových bloků (modulů), které lze vícenásobně a nezávisle použít v různých částech aplikací je základem aspektového programování, moduly bývají označovány za „aspekty“.

Aspektový přístup

- Nejdříve se vyhledají nezávislé „crosscutting concerns“ – například opakující se transakce nad databázemi, jejich logování, bezpečnostní procedury apod.
- Pak se naprogramují kódy jednotlivých aspektů a tyto kódy se připojí k odpovídajícím modulům technikou „weaving“ – spřádání. Teprve poté se provede případná kompilace.
- Vzniklý modul obsahuje jak obchodní logiku, tak programové kódy aspektů.
- Nejznámějším programovacím prostředkem pro realizaci aspektů je jazyk AspectJ, který vyvinul Georg Kiczales et al. ve vývojovém středisku firmy Xerox.
- Technika v případě AspectJ je založena na tom, že k běžícímu programu může být v určitém bodě připojena určitá modifikace „advice“ (filtr), která běh programu změní a vyvolá kód aspektu.

Převody dat

- Změny struktur stávajících dat do struktur nového systému
  - Jiné struktury tabulek
- Odstranění duplicit a chybných dat ze starého systému
  - Šance získat správná data
- Doplnění chybějících polí nebo údajů
  - Zpravidla ručně – velké problémy se zdroji
- Kdo provádí:
  - Koncoví uživatelé pod vedením klíčových uživatelů
  - Oddělení IT (automatizace převodu)
- Rizika:
  - Dodavatel nového systému nemá jasno o vztazích mezi daty (závažná chyba návrhu systému)
  - Oddělení IT má málo zdrojů
  - Koncoví uživatelů nemají čas na kontrolu
  - Nejsou jasné vazby na části IS, které nepodléhají změně
- Důsledek:
  - Může dojít k odložení náběhu

Akceptační /Integrační testy

- Probíhají za účasti celého týmu
- Účel:
  - Odzkoušet integritu systému (logistika a účetnictví, …)
  - Akceptace kritických funkcí a jejich vazeb
- Změnová past
  - Zpravidla teď se objeví další požadavky na funkcionalitu (úloha manažéra projektu)
  - Zavedení nových funkcí vyvolá chyby v již odzkoušených modulech
  - Oprava chyb vyvolá sekundární chyby
  - Dodavatel v dobré vůli provádí změny o své újmě
  - Nejsou k dispozici všichni klíčoví uživatelé k odzkoušení integrace
- Absolutní nutnost podrobně dokumentovat změny
- Nutnost iterativních postupů

Školení uživatelů

- Dva typy školení:
  - Train the trainer – jsou zaškoleni klíčoví uživatelé, kteří školí zbytek
  - Plné školení - provádí dodavatel se svých prostorách nebo u odběratele
- Změnová past:
  - I zde se projeví naplno požadavky konečných uživatelů na změny
- Nutnost dokonalé komunikace a projektového marketingu
- Nutnost oslovit pozitivně tvůrce veřejného mínění ve firmě
- Dokumentace
  - Dodavatelská – zpravidla nestačí, doplňují klíčoví uživatelé
  - Vlastní – dodavatel dodá jen systémovou dokumentaci, zbytek je úkolem odběratele (cenové dopady)

Náběh nového systému

- Dva typy náběhu:
  - Vše naráz „big bang“ - zpravidla u změn IS, které jsou značně provázané
  - S duplicitou – vyžaduje značné zdroje, menší riziko problému
- Rozhodnutí:
  - S dostatečným předstihem přijímá top management firmy
  - Nutnost stanovení „point of no return“
- Termínová past:
  - Závažné změny se provádějí na konci finančního období (fixní termín)
  - Rizika:
    - Uživatelé nejsou zaškoleni
    - Data nejsou připravena
    - Není dostatečně odzkoušena oblast kritických funkcí

Děkuji za pozornost.

Otázky?