Luboslav Lacko

Visual Studio 2005 Team System

VISUAL STUDIO 2005 TEAM SYSTEM

Luboslav Lacko

Autor je popredným slovenským odborníkom na databázové technológie a programovanie s viac než desaťročnou praxou vo vývoji databázových aplikácií. Pracuje vo Vojenskom technickom ústave v Liptovskom Mikuláši a pôsobí ako školiteľ a konzultant. Publikuje články predovšetkým o programovaní, databázach a data miningu; je autorom niekoľkých kníh, napríklad Business Intelligence na SQL Serveru 2005, ASP.NET 2.0 hotová Řešení.

OBSAH

KAPITOLA 1: VISUAL STUDIO 2005 TEAM SYSTEM

Motto 1:

„Riadiť a koordinovať prácu vývojárov a programátorov je rovnaké ako pásť mačky“

Motto 2:

„Babylónská rybka,“ odříkával tiše Stopařův průvodce po Galaxii, *„je malá, žlutá, trochu se podobá pijavici. Je to snad nejpodivnější tvor ve vesmíru. Živí se myšlenkovou energií, přijímanou nikoli od svého nositele, nýbrž z okolí. Má schopnost absorbovat a zpracovávat na živiny veškeré podvědomé frekvence této myšlenkové energie. Do mysli nositele vypouští své exkrementy v podobě telepatické matrice, vzniklé spojením vědomých myšlenkových frekvencí s nervovými signály přijímanými z řečových center mozku, který je vysílá. Praktický důsledek toho všeho je, že když si strčíte babylónskou rybku do ucha, okamžitě rozumíte čemukoliv, ať je to řečeno jakoukoli formou jazyka. Řečová schémata, která skutečně slyšíte, se dekódují podle myšlenkové matrice, kterou do vaší mysli uložila babylónská rybka.*

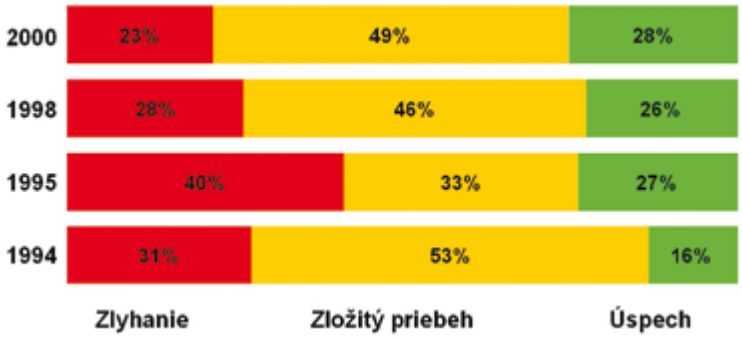
Nuž niečo podobné by sme potrebovali aj v oblasti programovacích jazykov a vývojárskych nástrojov hlavne pre tímovú spoluprácu. Napriek tomu, že je možné formulovať požiadavky na riadenie kvality procesu vývoja a tento proces dokonca do určitej miery štandardizovať, každý projekt vývoja softvéru je svojím spôsobom unikátny.

Každý väčší softvérový projekt je v súčasnosti už tímovou záležitosťou a k jeho úspešnosti a v neposlednom rade aj efektívnosti prispieva aj spôsob organizovania tímovej spolupráce. Vo väčšine prípadov vývojárske tímy dnes pracujú na svojich projektoch vo vývojárskych laboratóriách. V tejto oblasti by si mohli vziať vývojári príklad z globálneho biznisu, pre ktorý nielenže už neexistujú obmedzenia na jedno pracovisko, ale môžeme smelo povedať, že takmer už neexistujú ani žiadne geografické hranice. Predstavme si víziu, kedy na jednom softvérovom projekte pracujú koordinovane projektoví manažéri, architekti, vývojári a testovací pracovníci, pričom každý pracuje na svojom pracovisku, ktoré môžu byť geograficky ľubovoľne rozmiestnené podľa toho kde má firma pobočky. Pracovníci môžu byť prideľovaní do virtuálnych tímov, bez toho aby museli dočasne meniť svoje pracovisko, nemusia sa fyzicky stretávať na operatívnych mítingoch, strácať čas nekonečným telefonovaním a ani mailovaním. Každý si dokáže predstaviť časové, finančné a tím aj konkurenčné výhody, ktoré by priniesla realizácia tohto zdanlivo nedostižného sna.

Serverové Team Foundation Server je možné vyskúšať bez inštalácie online na adrese www.teamsystem.cz

Aspekty tímového vývoja

Úspech pri vývoji aplikácií nie je ani zďaleka takou bežnou záležitosťou, ako by sme sa na prvý pohľad mohli domnievať. Pomerne často dochádza v priebehu vývoja k problémom, ktoré nezriedka vedú dokonca k zlyhaniu celého projektu, prípadne k značnému posunu termínu dokončenia.



„Osud" približne 30 000 projektov vyvíjaných americkými firmami Zdroj: The Standish Group

International Inc., 2000–2004 Research Reports

Pokúsme sa zamyslieť nad najčastejšími príčinami zlyhania projektov. Tieto môžu byť na rôznych úrovniach hierarchie. Buď hierarchie architektúry projektu, alebo hierarchie riešiteľského tímu a jeho podnikového zázemia. Na najvyššej úrovni, ktorú môže ovplyvniť len „top manažment" je spravidla hlavným problémom chýbajúca podpora manažmentu a potenciálne nezahrnutie kľúčových hráčov, napríklad niektorých firemných oddelení, ktorých práca s projektom súvisí len nepriamo a podobne. Postupne ako sa presúvame do nižších úrovní podnikovej hierarchii môže byť príčinou neúspechu chýbajúca alebo zložitá komunikácia medzi vývojom a prevádzkou. Postupne ako klesáme v podnikovej hierarchii počet príčin narastá čoraz viac, no nemusia byť až také závažné. Využívanie tímovej spolupráce v oddelení vývoja považujeme za úplnú samozrejmosť, no aj na úrovni vývojových tímov sa vyskytujú rôzne problémy, napríklad nejasné ciele, zlá koncepcia, zmätený prístup, nejasný rozsah projektu a rozdelenie zodpovednosti, nedostatok alebo nevhodná infraštruktúra zdrojov, problémy s komunikáciou alebo osobnostné problémy. Ak to zhrnieme, väčšina problémov pri tímovom vývoji projektov sa viaže nie na technológie ale skôr na ľudský faktor, čiže ľudovo povedané „problém je medzi stoličkou a klávesnicou". S ľudským faktorom sú ako príčina neúspechu úzko spojené aj zle nastavené procesy. Projektoví manažéri sa väčšinou neúspešne snažia zaceľovať medzery vznikajúce medzi ľuďmi a procesmi v etape vývoja

Typické zloženie tímu pracujúcom na softvérovom projekte

Ak sa na príčiny potenciálnych alebo aj reálnych neúspechov opýtame samotných vývojárov, najčastejšie budú oscilovať odpovede typu: "Vývoj softvéru je veľmi obtiažny...", "Náš tím je veľmi rozptýlený a špecializovaný...", "Dnešné vývojárske nástroje nie sú dostatočne integrované...", "Potrebovali by sme predvídať postup našich projektov...", "Sme špeciálny prípad, máme svoje špecifiká, známe metodiky nám nevyhovujú...". Posledne citovaná odpoveď by sa v mnohých prípadoch dala parafrázovať do zrozumiteľného jazyka aj takto: „Sme iní ako tí ostatní... úspešní...".

Ak si pozrieme graf znázorňujúci pomer úspechu a neúspechu pri vývoji softvérových projektov, zistíme, že percento úspešnosti projektov postupne rastie. Príčin tohto pozitívneho trendu môže byť niekoľko. Do praxe sú uvádzané čoraz efektívnejšie metodiky nielen vývoja ale aj tímovej spolupráce. Moderné tímy pracujúce na softvérových projektoch sú v porovnaní s nedávnou minulosťou viac špecializované, viac sa orientujú na predmet podnikania. Možnosti tímovej spolupráce v rámci firemných intranetov, alebo globálneho Internetu umožňujú vhodnú geografickú distribúciu vývojárskeho tímu. Pri vývoji projektov veľkých medzinárodných korporácií tak čoraz viac vstupuje do hry obrovský potenciál vývojárov z Indie, Pakistanu, krajín Východnej Európy, Ukrajiny, Balkánu a Ruska. Je tu aj pomerne významná kladná spätná väzba, spôsobujúca, že k zvýšenému percentu úspešnosti nepochybne prispieva aj výrazné skrátenie cyklu vývoja.

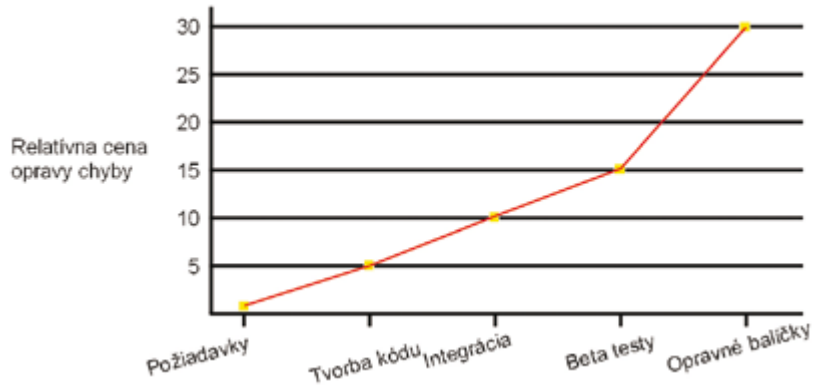
Všetky moderné vývojárske nástroje akceptujú a podporujú tímovú spoluprácu pri vývoji softvérových projektov ako jeden z hlavých pilierov svojej koncepcie. Používanie týchto nástrojov už od počiatku životného cyklu softvérovej aplikácie, teda už od procesu návrhu koncepcie a modelovania umožňuje zvýšiť predvídateľnosť procesu vývoja a to najmä zviditeľnením a zvýraznením údajov a informácií vedúcim k rôznym kľúčovým rozhodnutiam v procese vývoja. Možnosti integrácie vývojárskych nástrojov pre jednotlivé fázy cyklu spolu s možnosťami plánovania a modelovania procesu vývoja zaistia priebežné zachovanie kvality a zvýšia efektívnosť a produktivitu procesu vývoja softvéru.

Jedna z možných schém procesu vývoja softvérového projektu

Firma už niekoľko rokov ponúka metodiku ako organizovať ľudí a projekty pre úspešné a efektívne plánovanie, vývoj a nasadenie technologických riešení pod názvom Microsoft Solution Framework (MSF) (*poznámka autora: nemýliť si s technologickou platformou Microsoft .NET Framework*). Metodika by mala zaistiť budúci obchodný úspech projektu alebo riešenia prostredníctvom zladenia obchodných požiadaviek s dostupnou technológiou. MSF pri správnom nasadení a využívaní môže zaručiť vysokú kvalitu riešenia ponímanú zo všetkých uhlov pohľadu zároveň so zrýchlením vývoja, znížením nákladov a minimalizáciou rizík.

Dôležité je, kedy sa v procese vývoja chyba odhalí a opraví. Nasledujúci diagram by sa dal stručne charakterizovať sloganom „čím neskôr, tým drahšie" a k tomu sa ešte žiada dodať „a to exponenciálne". Takže prípadný omyl, alebo hoc aj závažnejšia nepresnosť, ktorú odhalíme už vo fáze formulovania požiadaviek nás vyjde oveľa lacnejšie ako chyba ktorú odhalíme až pri beta testovaní, alebo nebodaj až v ostrej prevádzke. Možno najmarkantnejšie si to dokážeme predstaviť napríklad pri vývoji softvéru pre riadiaci počítač osobného automobilu. Ak chybu odhalíme trebárs vo fáze kódovania, tak ju jednoducho

opravíme. Chyba odhalená v etape beta testov už znamená preprogramovanie väčšej, alebo menšej série čipov. No a najhoršie čo si môžeme predstaviť je sťahovanie už predaných automobilov do servisu kôli oprave chyby v softvéri riadiaceho počítača. Náklady s tým spojené si každý dokáže živo predstaviť



Cena opravy chyby závisí od toho kedy sa odhalí a opraví

Metodika pre vývoj softvéru

Pri návrhu a vývoji rozsiahlejších aplikácií je dôležitá aj určitá metodika, najlepšie taká, ktorá nám umožňuje uľahčenie a zjednodušenie vývoja a vnesie nám do nášho projektu určitý systém a poriadok. Pri úspešnom implementovaní dobrej metodiky spravidla dokážeme etapu vývoja aplikácie výrazne skrátiť, čo nám prinesie jednak úsporu nákladov a jednak pomôže zvýšiť konkurencieschopnosť nášho projektu. Veď história si spravidla pamätá len tých prvých. Šancu presadiť sa majú pochopiteľne aj tí ostatní (kvalita, invenčnosť), no platnosť ľudovej múdrosti: „...kto skôr príde, ten skôr melie..." je nespochybniteľná.

Hľadanie inšpirácie v prírode nie je ničím novým a k tejto metóde sa okrem konštruktérov lodí a lietadiel a rôznych mechanizmov uchyľujú aj informatici. A to nielen pri konštrukcii robotov a rôznych teórii neurónov a agentov, ale aj pri vývoji softvéru. V tejto oblasti sa okrem objektovo orientovaného programovania nechávajú informatici motivovať najmä Darwinovou evolučnou teóriou. Takto vznikli nielen evolučné algoritmy, ktoré patria v súčasnosti medzi aktuálnu problematiku informatiky a numerickej matematiky, ale aj evolučné metodiky pre vývoj softvéru. Správna metodika je dôležitá nielen v etape vývoja. Každý projekt je potrebné udržiavať a inovovať. V tejto etape sú potom pojmy ako prehľadnosť, dobrá dokumentácia, štruktúrovanosť a modulárnosť prakticky na nezaplatenie.

Poznámka autora: Je potrebné striktne odlíšiť evolučnú metodiku a evolučné algoritmy. Tieto algoritmy využívajú výpočtové modely evolučných procesov. Najdôležitejšie a najpoužívanejšie sú: genetické algoritmy, evolučné programovanie, evolučné stratégie, klasifikačné systémy a genetické programovanie. Všetky algoritmy sú založené na koncepte simulácie evolúcie jednotlivých štruktúr pomocou procesov ako sú selekcia, mutácia a reprodukcia. Tieto procesy sú závislé na charaktere jednotlivých štruktúr, ktorý je definovaný ako prostredie. Evolučné algoritmy patria skôr do pôsobnosti matematiky.

Aby sme si naplno uvedomili výhody metodiky MSF preferujúcej iteračný evolučný postup vývoja softvéru, predstavíme najskôr iné metodiky iných modelov

Metóda „veľkého tresku" (Big-Bang)
Pri použití tejto metodiky, ktorá vlastne ani metodikou nie je sa celý projekt rodí živelne, na rôznych úrovniach, spravidla nekoncepčne. Pre začiatok vývoja týmto spôsobom sú spravidla dva podnety. Buď sa v softvérovej firme „zrodí vízia" a vtedy to môže skončiť ako v lotérii. Občas obrovským úspechom, no častejšie s pomerne veľkými problémami. Druhým podnetom je požiadavka zákazníka na vývoj softvérového systému, kedy dodávateľská firma v snahe získať zákazku tvrdí, že takýto systém má už pripravený „na spadnutie". Po počiatočných fázach, ktoré sú spravidla sprevádzané aj neúspechmi sa začne pri vývoji projektu viac dbať na metodiku a vývojársky tím sa prikloní k iným koncepčnejším modelom.

Vodopádový model
Vodopádový model je odvodený podľa skúseností z iných inžinierskych procesov a je jedným z najdlhšie používaných modelov pre vývoj softvérových systémov. Ak si pozrieme schému modelu, kde jednotlivé fázy vývoja na seba kaskádovito nadväzujú, nebude nám pôvod názvu „Vodopádový model" nijakou záhadou Striktné oddelenie jednotlivých časti vývoja do samostatných fáz umožňuje sprehľadniť proces vývoja systému. Prechod do novej fázy vývoja je možný len po úplnom dokončení prác týkajúcich sa fázy predchádzajúcej. V opačnom prípade by bolo potrebné dodatočne sa vrátiť do predchádzajúcej fázy.
Vodopádový model pozostáva z niekoľkých fáz.

Vo fáze **analýzy požiadaviek**: analytici, alebo vývojársky tím vykonajú analýzu cieľov projektu a jeho požadovanej funkcionality. Ak sa jedná o softvér „na mieru" určený pre konkrétneho zákazníka, analýza prebehne v spolupráci so zákazníkom. Výsledky analýzy sú podrobne zdokumentované. V nadväznosti na výsledky analýzy sa vo fáze **návrhu architektúry** vytvoria konceptuálne a logické modely, ktoré vytvárajú celkovú architektúru systému **Podrobný návrh** sa týka najmä návrhu funkcií softvérového systému a spôsobu komunikácie. Po dosiaľ uvedených fázach, ktoré by sme mohli nazvať aj prípravné nasleduje fáza implementačná, čiže **kódovanie** a **testovanie** softvérových modulov. Po autonómnom otestovaní jednotlivých modulov sa tieto modely integrujú a následne t**estujú ako celý systém**. Potom môže nasledovať podľa povahy projektu jeho distribúcia alebo v prípade softvéru „na mieru" jeho odovzdanie

konkrétnemu zákazníkovi. Ani po tomto zdanlivo záverečnom úkone sa životné púť softvérového projektu nekončí. U zákazníka prebehne akceptačné testovanie, buď pred nasadením do plnej prevádzky, alebo niekedy aj počas nej. Počas celej doby životnosti softvéru je potrebné zaistiť jeho údržbu, napríklad odstraňovanie chýb, ktoré neboli objavené pri vývoji, vylepšovanie modulov systému a pridávanie nových funkcií. Rozhranie medzi jednotlivými fázami je však v praxi dosť neostré. Neostrým ho robia čiastočne aj chyby neodhalené v patričnej fáze, ktoré sa potom presúvajú do fáz následných.

Najväčšou výhodou vodopádového modelu je prehľadnosť postupu vývojových prác, nakoľko sú presne definované hranice medzi jeho jednotlivými fázami. Nevýhodou je značná miera abstrakcie na rôznych stupňoch, takže pre väčšie projekty jeho využitie prakticky nepripadá do úvahy. Nakoľko softvér je produktom s jednou z najvyšších mier abstrakcie je nevyhnutné, aby jeho tvorba prebiehala nie v jednosmerne naviazaných fázach, ale vo viacerých iteračných cykloch

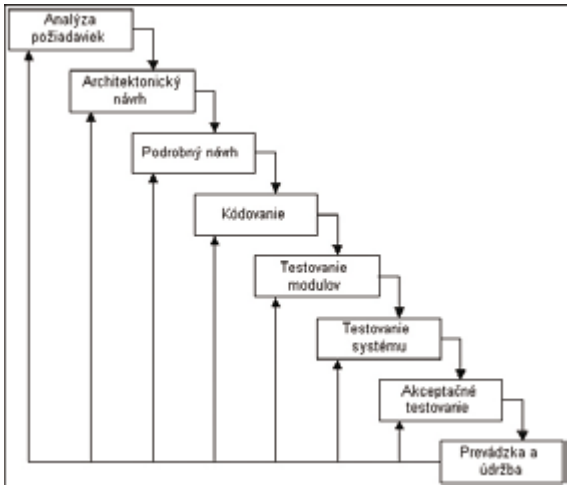


Schéma vodopádového modelu

„V" model
Na základe rozboru nevýhod vodopádového modelu, n ktoré sme poukázali v závere predchádzajúceho odstavca bol odvodený „V" model. Tento model už v sebe zahŕňa súvislosti medzi testovaním, analýzou a návrhom. Model má názov na základe usporiadania jeho blokov do tvaru písmena V, kde na ľavej strane sú fázy analýzy a návrhu a na pravej strane fázy testovania. Blok fázy kódovania je v strede. Ľavú stranu schémy môžeme nazvať vetvou vývoja a pravú stranu vetvou testovania a overovania systému. Čiarkované prepojenie signalizuje, že ak sa vo vetve overovania odhalia chyby, je potrebné vrátiť sa do vetvy vývoja na príslušnej úrovni.

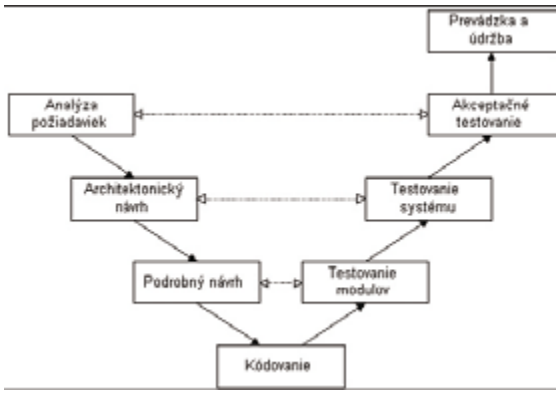


Schéma „V" modelu

Evolučný model

Vodopádový model aj jeho vylepšená „V" verzia poskytujú síce exaktnú metodiku, ktorá však na základe špecifických vlastností a vzťahov medzi vývojom softvéru a jeho funkcionalitou neposkytuje optimálne výsledky. Medzi pojmy funkcionalita softvéru a splnenie požiadaviek zákazníka totiž môžeme vložiť znamienko totožnosti. A je to práve snaha o splnenie požiadaviek zákazníka, ktorá vnáša do systému širokú mieru variability a aj určitú nestabilitu. S čím bol zákazník spokojný včera, nemusí byť dnes, jeho kolektívne myšlienkové procesy zákonite vedú k progresu a tento si vynucuje zmeny. Evolučný model odstraňuje presné definované hranice medzi jednotlivými fázami vývoja, keďže ani požiadavky zákazníka nie sú presné. Jedným z východísk z naznačenej situácie je vytvorenie čo možno najviac variantných prototypov, ktoré sa následne prekonzultujú so zákazníkmi. Prvotný prototyp podobne ako u vodopádového modelu vytvoríme na základe prvotnej abstraktnej špecifikácie, ktorá je výsledkom jednaní medzí analytikmi a zákazníkmi. Špecifikácia postupnými iteráciami spresňuje, pričom vstupom do každej iterácie sú aj vstupy od zákazníka na základe konfrontácie jeho požiadaviek s našim prototypom.

Podľa spôsobu vytvárania prototypov môžeme tento proces rozdeliť na dva typy:

– **Prieskumné prototypovanie** začína od častí systému, ktoré sú už v etape analýzy nejlepšie pochopené. Ďalšie časti systému sa do prototypov zahŕňajú po konzultáciách so zákazníkom. Takto postupne spresňujeme používateľské požiadavky

– **Vytváranie prototypov „na zahodenie"** sa zameriava na hlavne na problematické časti, ktoré nie je možné v etape analýzy a prvotného návrhu presne špecifikovať. Zpomedzi viacerých prototypov si nakoniec vyberieme taký prototyp, ktorý najlepšie spĺňa požiadavky. Takto sa dopracujeme buď k poslednej verzii, alebo k verzii, ktorá sa následne použije ako základ vo vývoji daného produktu, ale možno už podľa iného modelu, ktorý napríklad zaručí vyššiu spoľahlivosť, alebo menšie nároky na obsluhu a údržbu. Vytváraním prototypov čiastkových modulov „na zahodenie" sa môžeme prispôsobovať zmenám požiadaviek, preto sa táto metóda využíva hlavne vtedy ak je na začiatku vývoja problém vytvoriť dostatočne presnú špecifikáciu, alebo odôvodnene predpokladáme, že sa požiadavky počas vývoja môžu výrazne meniť. Určitou nevýhodou je neprehľadnosť procesu vývoja, pričom počet iterácií potrebných na dokončenie systému nie je vopred známy. To, že postup prác na systéme určuje vlastne zákazník prináša riziko zle štruktúrovaného výsledného produktu Zlá architektúra systému môže nezriedka vývoj sťažiť a predražiť.

Evolučný model možno efektívne uplatniť len pri vytváraní relatívne malých systémov, systémov s obmedzenou životnosťou alebo častí väčších systémov bez presnej špecifikácie. V prípade malých systémov môže byť požiadavka na zmenu systému v krajnom prípade riešená aj vytvorením celého systému odznova. U väčších systémov môžeme evolučnú metodiku využiť hlavne pri návrhu používateľského rozhrania.

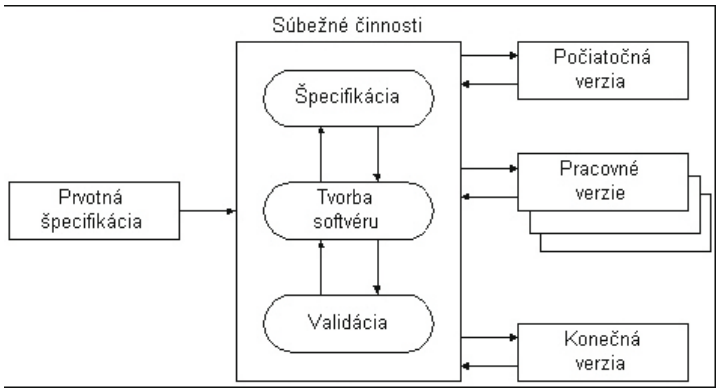
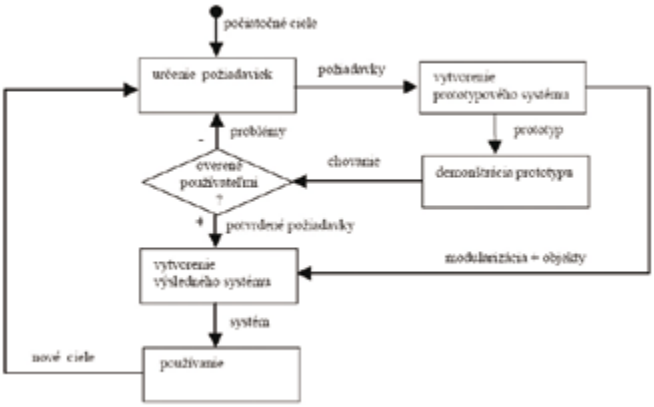


Schéma evolučného modelu



Iteračný prototypový evolučný model

Špirálový model
Na základe rozboru výhod a nevýhod vodopádového a evolučného modelu vznikol špirálový model.

Grafickým znázornením modelu je špirála, u ktorej každý závit zodpovedá jednej fáze vývoja. Cykly naprieč špirálou v smere od stredu k okraju zodpovedajú fázam jako ich poznáme z vodopádového modelu. Nahradenie kaskádového radenia špirálovým usporiadaním prináša možnosť iterácie. Navyše v takomto modeli nie sú presne jednotlivé fázy presne ohraničené. Jednotlivé cykly (závity špirály) pozostávajú z niekoľkých segmentov
– určenie cieľa
– stanovenie rizík a možnosti ich eliminácie
– vývoj a testovanie
– plánovanie

V segmente plánovania sa rozhoduje, či je potrebný ďalší cyklus iterácie. Výhodou špirálového modelu je jeho prehľadnosť vďaka rozdeleniu vývoja do jednotlivých fáz. Zároveň ale umožňuje viackrát sa vrátiť k jednotlivým fázam na základe požiadaviek zákazníka. Najväčšou výhodou špirálového modelu však je zahrnutie rizík spôsobených nedostatkom presných informácia ich eliminácia. Špirálový model je aj značne variantný, nakoľko v jednotlivých fázach vývoja môžeme použiť rôzne typy modelovania.

Metodika „Microsoft Solution Framework“

Uplynulo snáď už veľa času od bývalej socialistickej „éry“, kedy sa o kvalite síce veľa hovorilo, ale v praxi to bol pomerne neznámy pojem. V súčasnosti už väčšina firiem uznáva a používa osvedčené systémy riadenia kvality. Mnohí programátori sa možno nad snahou zapojiť vývoj do systému riadenia kvality pousmejú, no ak si vypočujeme aj názor druhej strany, neraz sa stalo, že manažér riadenia kvality „objavil“ vo firme skupinku mierne spustnutých, spravidla o nejakom abstraktnom probléme živo diskutujúcich indivíduí – programátorov. Táto situácia spravidla vyústi do situácie, kedy bude proces vývoja softvéru zaradený ako štandardný proces v podnikovom riadení kvality, Metodika MSF sa počas

25 ročných skúseností Microsoftu s vývojom softvéru, či už sa jedná o interný vývoj, teda práce na vývoji softvérových produktov Microsoftu, alebo skúsenosti vývojárov využívajúcich vývojárske nástroje, ktoré Microsoft dodáva, rozčlenila na dve vetvy

MSF for Agile Software Development
Jednoduchšia, používateľsky prívetivejšia metodika, ktorá sa snaží akceptovať individuálny prístup a kreativitu programátorov. Vhodnejšia je skôr pre menšie tímy, prípadne je možné veľký realizačný tím rozčleniť na viac menších „podtímov", ktoré samostatne riešia určitú časť projektu, napríklad niektorú relatívne uzavretú vrstvu architektúry, čiastkovú aplikáciu, ktorá bude súčasťou väčšieho projektu a podobne. Metodika vychádza z osvedčených scenárov. Preferuje požiadavky na kvalitu procesu vývoja a zároveň eliminuje riziká. Postup a mieru úspešnosti projektu sa posudzuje na základe rôznych kritérií, napríklad časového postupu, počtu chýb a podobne. Okrem eleminácie rizík je jednou z hlavných výhod, ktorú prinesie zavedenie a akceptovanie tejto metodiky je, že do kontextu projektu je zaradené aj testovanie.

MSF for CMMI Process Improvement
Skratka CMMI znamená „Capability Maturity Model Integration". Jedná sa o štandard metodiky pre vývojárske projekty, ktorý bol definovaný inštitútom Carnegie Mellon Software Engineering Institute. Ak by sme chceli túto metodiku zaradiť v kontexte filozofie metodiky Microsoftu, mohli by sme ju charakterizovať ako určitú nadmnožinu metodiky MSF for Agile. Táto metodika vychádza z eliminácie rizík a snaží sa maximálne klásť dôraz na proces. Nakoľko proces vývoja vyžaduje aj určitú mieru kreativity, ktorá je často v dosť príkrom rozpore s požiadavkou na dôsledné dodržiavanie procesných náležitostí, nemusí dôsledné aplikovanie CMMI metodiky vždy proces vývoja zefektívniť

Základnými piliermi metodiky MSF sú
- evolučný iteračný proces vývoja, testovania a zavádzania nových verzií
- aktívne dokumenty
- matica kompromisov

Iteračný proces
Ak sledujete proces beta testovania a zavádzania nových produktov Microsoftu na trh, napríklad v dobe prípravy tohto materiálu finišovalo beta testovanie operačného systému Windows Vista a kancelárskeho Balíka MS Office 2007 viete, že najskôr sa objaví niekoľko, čoraz vyzretejších beta verzií (BETA1, 2, 3...). Po beta verziách nasleduje verzie RC a RTM a následne sa produkt objaví na trhu v prvej finálnej verzii. Každá verzia je dôsledne uzavretá (k určitému dátumu) a zdokumentovaná. Samozrejme sa predpokladá, že s postupným príchodom verzií bude produkt čoraz viac vyzretejší. Dôsledné uzatváranie jednotlivých verzií je potrebné hlavne preto, lebo takto máme vždy aktuálnu verziu, ktorú môžeme predvádzať zákazníkom.

Aktívne dokumenty
Filozofia aktívnych dokumentov predpokladá, že všetky dokumenty súvisiace s projektom budú „živé", to znamená, že v celom procese nebudú uzavreté, ale sa do nich budú riadene a dynamicky zaznamenávať potrebné zmeny. Všetky zmeny v aktívnych dokumentoch je potrebné dokumentovať a dokumenty presne verziovať.

Matica kompromisov
Maticu kompromisom si môžeme predstaviť ako dvojrozmernú tabuľku. V riadkoch máme jednotlivé kritéria a atribúty požadované pre softvérový projekt. Typicky sú to
- zdroje
- termíny
- funkčnosť
- kvalita

Úlohou matice kompromisov je nájsť vhodný a prijateľný kompromis medzi zdrojmi, ktoré má vývojársky tím k dispozíciami, termínmi, ktoré vyplývajú zo zmlúv alebo jednoducho z požiadavky aby produkt prišiel na trh skôr ako produkt od konkurencie a funkčnosťou aplikácií. Pre niektorý atribút sú pravidlá pevne dané, iné si zvolíme a na základe toho upravíme požiadavky na zbývajúci atribút. Napríklad máme fixné zo zmluvy vyplývajúce termíny. Na základe toho vo firme pridelíme k jednotlivým projektom zdroje, napríklad členov tímu, kapacity testovačov a podobne.

Všimnite si štvrtý atribút – teda kvalitu. Podľa teórie síce do matice kompromisov patrí, čiže ak máme napríklad obmedzené zdroje a šibeničné termíny, aj tak by sme dokázali vytvoriť aplikáciu, u ktorej by však kvalita trochu pokuľhávala. Nakoľko však už staré dobré časy socialistickej „nekvality" máme bezpečne za sebou, v otázke kvality sa nepripúšťajú žiadne kompromisy, preto pre ďalšie úvahy toto kritérium z matice kompromisov jednoducho vylúčime a kvalitu budeme požadovať konštantnú. Radšej v núdzi obmedzíme funkcionalitu, to znamená, že aplikácia bude mať menej funkciuí, ale kvalitných.

Na základe týchto dvoch atribútov potom musíme upraviť požiadavky na funkcionalitu aplikácie. Microsoft vo svojich metodikách uprednostňuje maticu kompromisu definovanú tak, že zdroje sú fixné, na základe toho zvolíme termíny a funkcionalita bude upravená

Maticu kompromisov dokonca môžeme zapísať ako vetu v prirodzenom jazyku podľa tejto šablóny:

Ak máme fixné _____,
môžeme si zvoliť _____,
a podľa toho musíme upraviť _____.

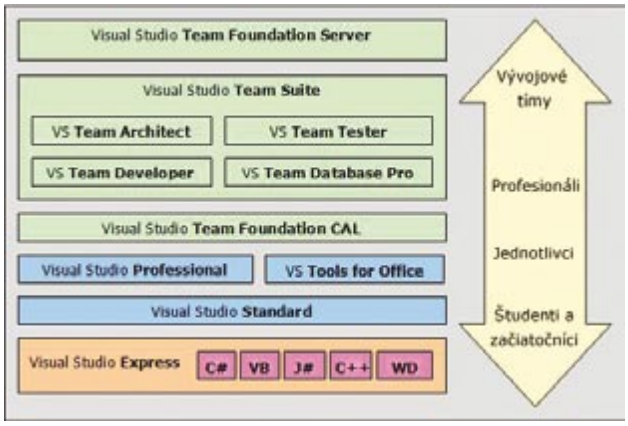
Matica kompromisov (zaškrtnuté sú typické kombinácie)

Napríklad
Ak máme fixné zdroje,
môžeme si zvoliť termíny,
a podľa toho musíme upraviť funkčnosť.

Architektúra a filozofia Team Foundation serveru vychádza z metodiky MSF.

Určenie verzií vývojového prostredia Visual Studio 2005

Uvedením Visual Studia 2005 sa výrazne rozšírilo portfólio spoločnosti Microsoft pre vývojárov. Na jednej strane sa posúvajú produkty firmy k najvyšším úrovniam tímovej spolupráce a riadenia životného cyklu aplikácií využitím najmodernejších technológií, na druhej strane sú po prvý krát dostupne edície vývojárskych produktov spoločnosti Microsoft úplne zadarmo.



Určenie jednotlivých verzií vývojového prostredia Visual Studio 2005

Visual Studio 2005 Team System

Vývojové prostredie Visual Studio 2005 vo verzii „Team System“ sa k realizácii vízií naznačených v predchádzajúcom odseku značne priblížilo. Umožňuje efektívnu podporu tímovej spolupráce naprieč celým procesom vývoja aplikácií počnúc modelovaním a návrhom architektúry až po testovanie a úpravy aplikácie podľa dodatočných požiadaviek zákazníkov.

Visual Studio 2005 Team System využíva klient – server architektúru, pričom na klientskej strane je na klientských vývojárskych počítačoch vývojové prostredie a ako serverová časť slúži Team Foundation Server. Hlavným cieľom tohto tandemu je na základe aplikovania vhodnej metodiky a sledovania priebehu projektu monitorovať, v akom štádiu sa projekt nachádza, aké sú komunikačné toky medzi jednotlivými členmi tímu navzájom a podobne. Takto môžu manažéri projektu získať vo vhodnej podobe dostatok informácií, aby mohli včas reagovať na potenciálne problémy, napríklad ak hrozí nesplnenie dôležitého termínu a podobne. Monitorovaním tímového vývoja sa do značnej miery zvýši prehľadnosť a predvídateľnosť celého procesu vývoja a tím aj jeho produktivita a efektívnosť integrácie

Vývojové prostredie Visual Studio má za sebou bohatú históriu od verzie 1.52, ktorá ešte podporovala vývoj pre operačný systém MSDOS, cez verzie 4.0, 5.0, 6.0 a verzie Visual Studio 2002, 2003 a 2005 založené na riadenom kóde a technologickej platforme .NET Framework. Na základe skúseností vývojom na týchto vývojárskych platformách vytvoril Microsoft dve metodiky zaoberajúce sa aspektom tímového vývoja „Microsoft Solution Framework Agile“ a „Microsoft Solution Framework for CMMI Level 3“. Metodiky riešia hlavne manažment rolí v procese cyklu vývoja aplikácií a opatovné použitie, výmenu a zdieľanie komponentov, ktoré sú základnými stavebnými kameňmi softvérových riešení. Microsoft pri návrhu podpory tímovej spolupráce na softvérových projektov „Team Systems“ využil aj skúsenosti z riešenia pre správu projektov. „Microsoft Project“

Vývojové prostredie Visual Studio 2005 vo verzii Team System umožňuje spoluprácu projektových manažérov, architektov, vývojárov a testovacích pracovníkov z jedného spoločného integrovaného používateľského prostredia.

Samozrejmosťou je podpora projektov využívajúcich SOA architektúru, čiže architektúru zameranú na služby.

Súčasné aplikácie sú veľmi komplexné a prepracované. Vývoj, to nie sú len programátori. Pre vývoj zložitých a komplexných projektov je potrebný zohraný tím, ktorý má jasne a efektívne rozdelené jednotlivé role. Musí byť jasná na jednej strane pôsobnosť a na druhej strane zodpovednosť rolí architekt aplikácie, správca projektu, programátor, tester a v neposlednom rade dokonca aj koncový používateľ. Najdôležitejšie v celom tíme je dobrá a riadená komunikácia, správne navrhnutý a dodržiavaný životný cyklus aplikácie, predvídateľnosť vývoja, ktorá vedie k opakovaným výsledkom a vysoká produktivita. Balík produktov Visual Studio 2005 Team System podporuje celý životný cyklus vývoja sw aplikácií – od návrhu špecifikácie po uvedenie do prevádzky. Pri vývoji v tíme pomôže:

- zvýšiť produktivitu (a tým znížiť náklady)
- zvýšiť predvídateľnosť (a tým znížiť riziká)

Veľmi dôležitou črtou je integrácia, tak aby jednotlivé nástroje, ktoré tvoria vývojové prostredie fungovali dohromady. Jednotlivé nástroje a časti riešenia je samozrejme možné obnoviť na novšiu verziu, prípadne nahradiť alebo modifikovať. Vývojové prostredie Visual Studio 2005 je vzhľadom na svoju históriu pomerne vyzretým produktom a v dobe písania tejto publikácie boli už aj relatívne dostupné informácie o ďalšej verzii, zatiaľ známej pod kódovým označením „Orcas“. To dáva manažérom zodpovedným za IT a vývoj možnosť stanoviť jasný plán rozvoja vybavenia oddelenia a naplánovať prípadné migrácie na nové verzie hlavne v nadväznosti na plán vývoja, údržby a šírenia aplikácie. Samozrejme neoddeliteľnou súčasťou rozvoja je plán rekvalifikácie a vzdelávania jednotlivých členov tímu

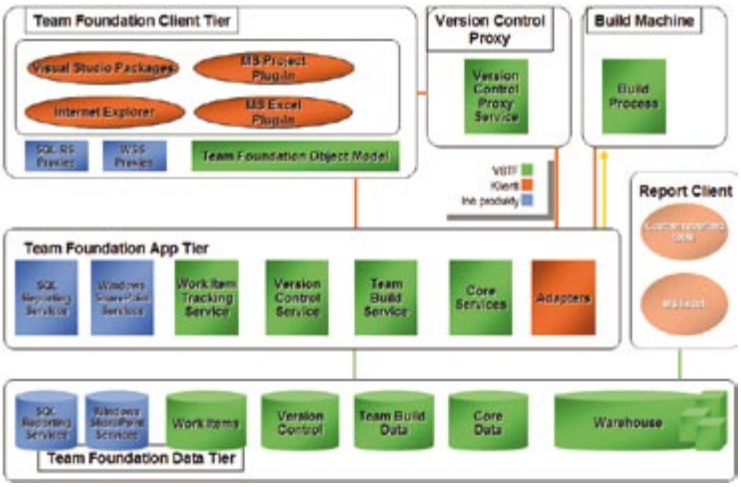
Architektúra Team System

Architektúra Visual Studio 2005 vo verzii „Team System“ je založená na viacerých vrstvách.

Dátová vrstva
Zoznamovanie s jednotlivými vrstvami architektúry začneme na úrovni dátovej vrstvy. Taká činnosť, ako je riadenie a koordinácia tímovej práce na projekte predpokladá možnosť bezpečného, spoľahlivého a operatívneho ukladania určitých objemov údajov. Preto sú spomínané údaje uložené v databáze pod správou databázového serveru. Nakoľko sa jedná o platformu Microsoftu, je pochopiteľné, že pre ukladanie a správu údajov slúži databázový server SQL Server 2005

Aplikačná vrstva
Tak ako sme u dátovej vrstvy skonštatovali, že využíva technológie Microsoftu, aj aplikačná vrstva je logicky založená na serverovej technológii ASP.NET stránok, ktoré bežia pod správou webového servera IIS (Internet Information Server)

Klientská vrstva
Služby aplikačnej vrstvy využívajú na svojich klientských počítačoch jednotliví členovia tímu, teda vývojári, architekti, manažéri a podobne. Podľa pracovného zaradenia využívajú jednotliví členovia tímu rôzne nástroje, napríklad vývojové prostredie, program Excel, prípadne aplikácia Microsoft Project. Komunikácia klientských aplikácií s aplikačnou vrstvou prebieha ako u klasických webových, alebo intranetových aplikácií, pričom sa do značnej miery využíva aj technológia webových služieb.



Koncepcia a architektúra Visual Studia 2005 „Team System“

Okrem troch vymenovaných hlavných vrstiev si na schéme všimnite aj ďalšie bloky. Konfigurácia IT prostredia pre tímový vývoj môže obsahovať aj ďalší server „Build Machine“ na ktorom sú umiestnené jednotlivé verzie vyvíjaných produktov. Keďže nové verzie vyvíjaného produktu sú vlastne hlavným produktom vývoja, „Build Machine“ môžeme prirovnať k akémusi skladu finálnych produktov určených na testovanie, prípadne na distribúciu. Pre rýchlejší prístup viacerých vzdialených klientov je možné do konfigurácie zaradiť proxy server „Version Control Proxy“. Technicky sa jedná o aplikáciu bežiacu pod IIS. Na schéme architektúry vpravo vidíme blok klientského prístupu k reportom. Pre klientský prístup k reportom môžeme použiť napríklad program MS Excel, alebo inú vhodnú, napríklad na mieru ušitú klientskú aplikáciu. Samozrejme nie každý blok (databázový server, aplikačný server, build machine...) musí bežať na samostatnom „železe“. V niektorých prípadoch môžu všetky serverové vrstvy bežať na jednom jedinom fyzickom serveri. Ak predsa len chceme viac alebo menej striktne od seba oddeliť jednotlivé architektonické vrstvy, veľmi zaujímavou možnosťou by mohla byť technológia virtualizácie s využitím virtuálnych serverov vytvorených a konsolidovaných pomocou produktu Virtual Server 2005.

Dátový sklad
V konfigurácii pre tímový vývoj musí byť aj dátový sklad, do ktorého sa zavádzajú údaje z jednotlivých databáz. Na dátový sklad sú napojené analytické služby. V nich je vytvorená kocka „Team System“

s viacerými dimenziami, napríklad časová dimenzia pre dátum, personálna dimenzia zainteresovaných osôb, teda členov tímu, „produktová dimenzia" zahrňujúca v tomto prípade buildy vyvíjaných aplikácií. Ďalšie dimenzie sa týkajú napríklad kvality vývoja, stavu jednotlivých projektov, ich revízií, ale trebárs aj dimenzia pre adresáre, v ktorých sa nachádzajú produkty vývoja. Výsledky analýz sú prístupne pomocou reportov generovaných prostredníctvom reportovacích zložieb napojených jednak na relačné databázy, na dátový sklad a taktiež aj na analytické služby. Fyzicky je dátový sklad realizovaný pod správou SQL Servera 2005 ako databáza TFSwarehouse.

Visual Studio 2005 Team System – prehľad verzií a vlastností:

Vývojové prostredie VSTS sa dodáva vo viacerých verziách, čo umožňuje bohatú škálovateľnosť a výber optimálnej verzie pre daný scenár nasadenia

Visual Studio 2005 Team Suite
Je najproduktívnejší integrovaný balík nástrojov pre vývoj všetkých typov programov, obsahujúci všetky dostupné technológie na podporu životného cyklu softvérových aplikácií. Nájdete tu produkty od projektového návrhu, architektúry, cez vývoj až po testovanie vrátane potrebnej infraštruktúry.

Visual Studio 2005 Team Edition for Database Professionals
Verzia pre databázových profesionálov poskytuje nástroje na správu zmien a zdrojových kódov, testovanie a nasadzovanie databázových riešení založených na SQL serveri. Aktívne zapája čisto databázového špecialistu do vývojového tímu.

Microsoft Visual Studio 2005 Team Edition for Software Architects *Verzia pre softvérových architektov*

Visual Studio 2005 Team Edition for Software Architects
Verzia určená pre softvérových architektov poskytuje nástroje pre vizuálny design service-oriented riešení a ich validáciu vzhľadom k aplikačnému prostrediu a infraštruktúre v ktorej budú nasadzované.

Microsoft Visual Studio 2005 Team Edition for Software Developers *Verzia pre vývojárov*

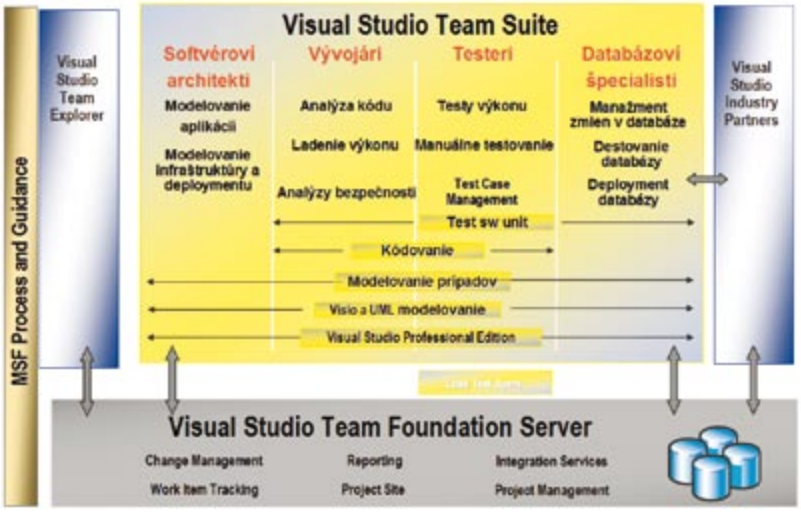
Visual Studio 2005 Team Edition for Software Developers
Team Edition for Software Developers ponúka pokročilé vývojové nástroje ktoré posúvajú kvalitu vývojárskej práce v tímoch na vyššiu úroveň bez nutnosti prebytočných úkonov vyžadovaných od vývojárov.

Microsoft Visual Studio 2005 Team Edition for Software Testers *Verzia pre testovanie softvéru*

Visual Studio 2005 Team Edition for Software Testers
Team Edition for Software Testers predstavuje balík nástrojov integrovaných do prostredia Visual Studio, ktoré pomáhajú vytvárať bezchybné aplikácie, testovať ich a zdieľať výsledky s celým tímom aj koncovým užívateľom.

Visual Studio 2005 Team Foundation Server
Serverový produkt pre spoluprácu, zaisťujúci čo najúčinnejšiu komunikáciu vo vnútro celého IT tímu a jednoduchú správu a sledovanie vývoja a stavu projektov. Kombinuje správu zdrojov, všetkých metadát a väzieb, reportingové a portálové služby do jedného homogénneho riešenia s overeným API. Tento produkt je možné vyskúšať si na www.teamsystem.cz .

Visual Studio 2005 Team Test Load Agent
Je dodatočný testovací nástroj pre testerov, ktorý umožňuje simulovať výrazne vyššie množstvo užívateľov (ako VS Tester Edícia) s väčšou presnosťou, výkonom a stabilitou webových aplikácií a serverov.



Koncepcia verzií Visual Studia 2005 „Team System"

Určenie jednotlivých verzií zobrazuje prehľadne nasledujúca tabuľka

Visual Studio 2005 Team Suite			
Team Edition for Software Architects	**Team Edition for Software Developers**	**Team Edition for Software Testers**	**Team Edition for Database Professionals**
Appplication Designer			
System Designer			
Logical Center Designer			
Deployment Designer			
	Code Profiler		
	Static code Analyzer		
	Dynamic code Analyzer		
		Load Testing	
		Manual Testing	
		Test Case Management	
	Code Coverage		
	Unit testing (C#, VB.NET)		
			Unit Testing (T-SQL)
			DB Rename Refactoring
			Offline Database Project
			Data Test Generator
			Data Compare
			DB Schema Compare
Visual Studio 2005 Professional			
Team Foundation Server CAL			

Visual Studio Team Foundation Server			
Change Management Reporting	Project Portal Project Management	Team Build Server Work Item Tracking	Integration Services

Určenie verzií Visual Studia 2005 „Team System"

Visual Studio 2005 je tesne integrované s ďalšími serverovými a podnikovými aplikáciami Microsoftu

- **SQL Server 2005** – Úložisko pre údaje a dokumenty potrebné pre tímovú spoluprácu. Reportovacie služby SQL Serveru sa využívajú pre generovanie výstupných reportov
- **Windows SharePoint Portal Services** – Tímový projektový portál.
- **Microsoft Project 2003** – Nástroj pre správu projektov
- **Microsoft Excel 2003** – Tabuľkový procesor pre vytváranie dokumentov
- **Internet Explorer** – Webový prehliadač pre prístup na portál a prezeranie reportov

Inštalácia Team Foundation Serveru

Inštalácia servera pre tímový vývoj je pomerne komplikovanou záležitosťou a je potrebné dôsledne postupovať podľa inštalačného manuálu a nainštalovať všetky potrebné produkty v stanovenom poradí a nezabudnúť na predpísané (alebo vyššie – najaktuálnejšie) verziu opravných balíčkov.

Minimálne hardvérové požiadavky pri dnešnom neustálom zvyšovaní výkonu moderných počítačov nie sú až také náročné. Spomeňme minimálne rozumné požiadavky: procesor minimálne 600 MHz, 512 MB RAM, niekoľko gigabajtov voľného miesta na disku a minimálne použiteľné rozlíšenie displeja 1024x768. Informáciu o optimálnej konfigurácii nájdeme napríklad na adrese http://blogs.msdn.com/bharry/archive/2005/10/24/484125.aspx,

Podporované sú operačné systémy:
Microsoft® Windows 2000 + Service Pack 4
Microsoft® Windows XP + Service Pack 2
Microsoft® Windows XP Professional x64 Edition
Microsoft® Windows Server™ 2003 + Service Pack 1
Microsoft® Windows Server™ 2003, x64 Edition
Microsoft® Windows Server™ 2003 R2
Microsoft® Windows Server™ 2003 R2
Microsoft® Windows Vista™

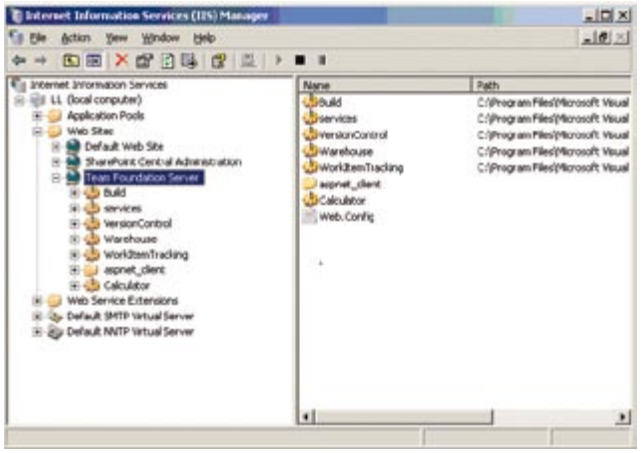
Uvádzame aj zoznam potrebného, prípadne voliteľného programového vybavenia potrebnú pre tímovú spoluprácu pod taktovkou Team Foundation Servera

Visual Studio 2005 Professional nebo tímová verzia
Na pracovnej stanici je potrebné mať nainštalované Visual Studio 2005 v edícii Professional alebo niektorú s verzií pre tímovú spoluprácu (Team Architect, Team Developer, Team Tester, Team Suite).

VS 2005 Team Explorer
Visual Studio 2005 Team Explorer je prídavný balíček do vývojového prostredia Visual Studio 2005, ktorý umožňuje prístup k Team Foundation Serveru a zobrazenie údajov ním spravovaných. Podobný balíček je k dispozícii aj pre programy Excel a Project 2003. Team Explorer je možné použiť aj vez Visual Studia, no v takom prípade nie je možné robiť tímové projekty

Aby sme lepšie pochopili konfiguráciu a princíp fungovania Team Foundation Serveru, je vhodné pozrieť si v administrátorských nástrojoch administrátorskú konzolu Internet Information Serveru. Team Foundation Server sa v tejto konzole javí ako virtuálne webové sídlo. V jednotlivých zložkách môžeme preskúmať webové služby pomocou ktorých pristupujú klientské aplikácie ku serverovým službám.

Podobne môžeme pomocou administrátorskej konzoly SQL serveru „SQL Server Management Studio" preskúmať štruktúru údajov uložených v databázach. Team Foundation Server pri svojej inštalácii vytvorí niekoľko databáz. Môžeme ich jednoducho identifikovať, pretože ich názvy začínajú na písmená „TFs"



Team Foundation Server beží pod IIS

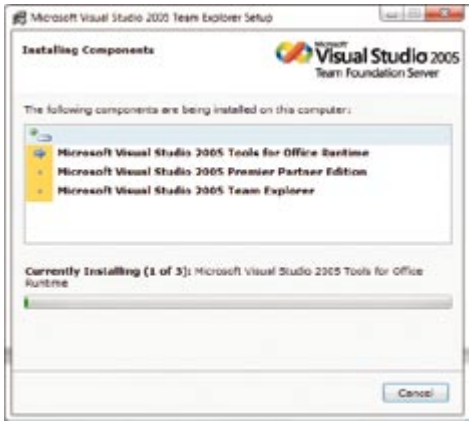
Inštalácia doplnku MS Visual Studio 2005 Team Explorer

Aby sme sa mohli z klientského vývojárskeho počítača na ktorom je nainštalované vývojové prostredie Visual Studio 2005 pripájať k Team Foundaion Serveru, je potrebné nainštalovať do vývojového prostredia doplnok Visual Studio 2005 Team Explorer.

Doplnok je možné stiahnuť napríklad na adrese
http://download.microsoft.com/download/2/a/d/2ad44873-8ccb-4a1b-9c0d-23224b3ba34c/VSTFClient.img

Team Explorer potrebujeme aj v prípade ak si chceme tímovú spoluprácu cez Team Foundation Server vyskúšať bez inštalácie online na adrese www.teamsystem.cz

Inštalácia doplnku MS Visual Studio 2005 Team Explorer

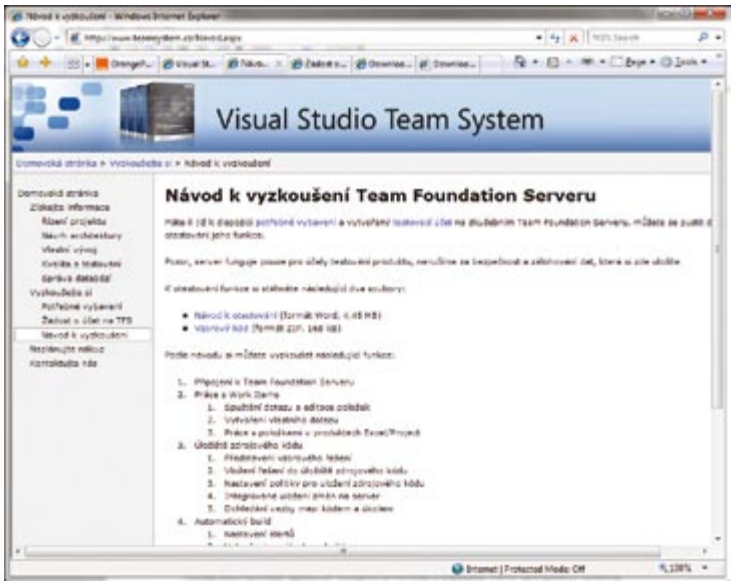


Priebeh inštalácie Team Explorera

Vyskúšanie Team Foundation Serveru

Nakoľko inštalácia Team Foundation Serveru nie je práve triviálnou záležitosťou a pre jej úspešnú realizáciu je potrebné mať nielen príslušné inštalačné média, ale hlavne veľa skúseností s inštaláciou a konfiguráciou jednotlivých komponentov, pripravil Microsoft možnosť vyskúšať si toto prostredie pre tímovú spoluprácu bez inštalácie online na adrese www.teamsystem.cz

Najskôr je potrebné vyplnením formulára na adrese http://www.teamsystem.cz/ZadostUcet.aspx požiadať o vytvorenie používateľského účtu na Team Foundation Serveri. Žiadateľ dostane podrobnejšie informácie mailom do dvoch pracovných dní. Na klientskom počítači je potrebné mať nainštalovanú verziu vývojového prostredia Visual Studio 2005 Professional Edition.



Stránka www.teamsystem.cz pre on line vyskúšanie Team Foundation Serveru

V prípade kladného vybavenia žiadosti dostane používateľ údaje k svojmu testovaciemu účtu v tvare

Portál – http://demo.teamsystem.cz/sites/xxx
Reporty – http://demo.teamsystem.cz/Reports/Pages/Folder.aspx?ItemPath=/xxx

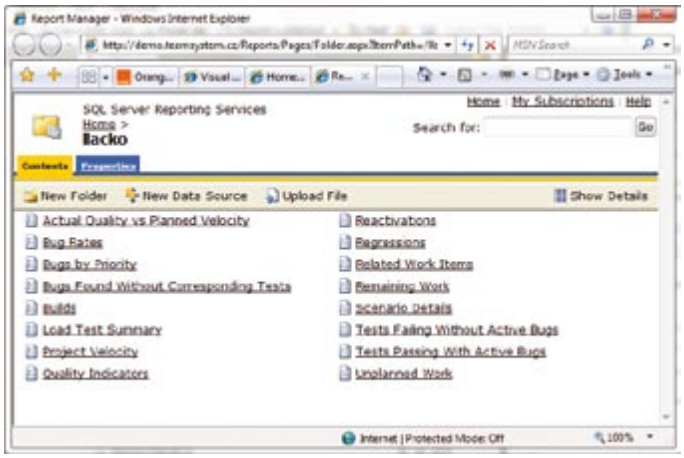
Web sluzba TFS (kontrola konektivity) – http://demo.teamsystem.cz:8080/services/v1.0/ServerStatus.asmx

Uživatelské jméno: TEAMSYSTEM\xxxx
Heslo: xxxx

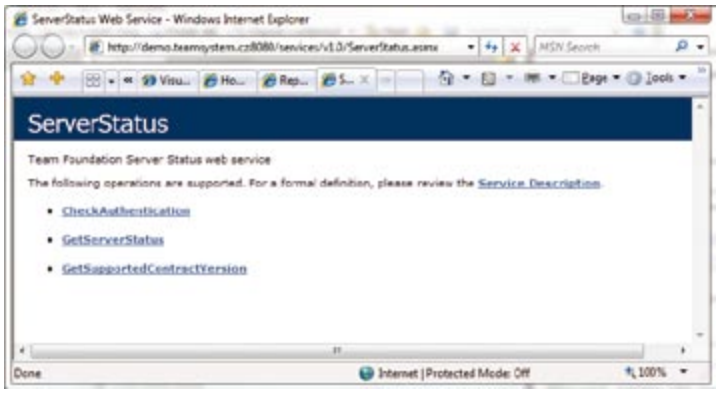
Po úspešnom prihlásení sa dostaneme na hlavnú stránku vývojárskeho portálu

Stránka testovacieho používateľského účtu

Taktiež si môžeme pozrieť stránku s reportami. Po zadaní príslušnej URL adresy sa zobrazí stránka Reoprtovacích služieb SQL Serveru 2005 so zoznamom reportov. Kliknutím na odkazy s názvami reportov môžeme jednotlivé reporty preskúmať

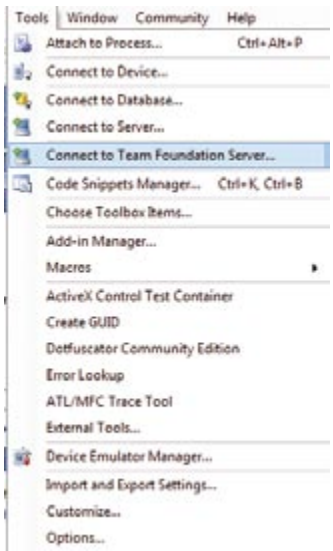


Stránka reportov testovacieho používateľského účtu



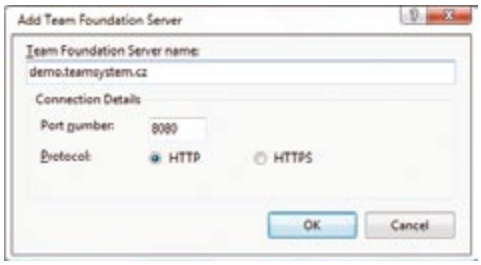
Stránka webovej služby

Pre pripojenie sa ku testovaciemu serveru z vývojového prostredia je potrebné mať nainštalovaný doplnok nainštalovaný doplnok Team Explorer.
Ak máme do vývojového prostredia nainštalovaný doplnok Team Explorer, pripojíme sa k testovaciemu serveru pomocou menu Tools/Connect to Team Foundation Server... :



Menu pre pripojenie sa k Team Foundation Serveru

Najskôr pridáme parametre pre pripojenie sa k Team Foundation serveru. Pre testovacie konto je adresa demo.teamsystem.cz



Pridanie pripojenia na Team Foundation Server

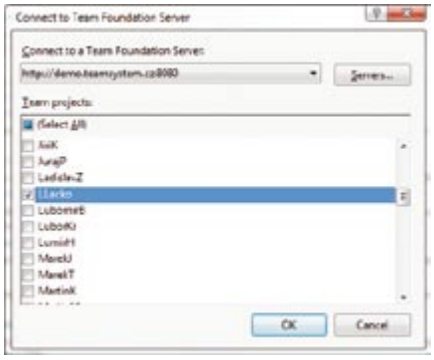
Po potvrdení URL adresy servera sa prihlásime pomocou pridelených prístupových privilégií.

Prihlásenie sa v vybranému Team Foundation Serveru

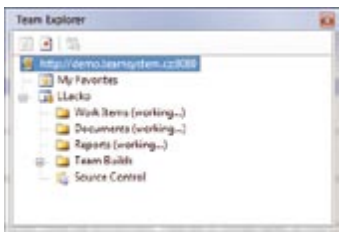
Po úspešnom pripojení sa na server bude jeho meno a URL adresa pridaná do zoznamu pripojení k serverom

Zoznam pripojení k Team Foundation Serveru

Po úspešnom pripojení sa k Team Foundation Serveru sa zobrazí zoznam projektov dostupných na serveri.



Pripojenie sa ku konkrétnemu projektu



Okno Team Explorera

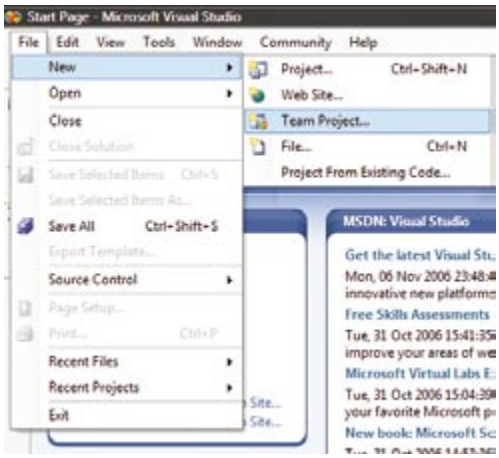
Vytvorenie tímového projektu

Pri vytváraní nového projektu je potrebné si najskôr zvoliť metodiku podľa ktorej bude projekt vytvorený ale najmä podľa filozofie ktorej bude projekt fungovať. V predchádzajúcich statiach boli detailne popísané nielen princípy, ale aj výhody a nevýhody metodológií

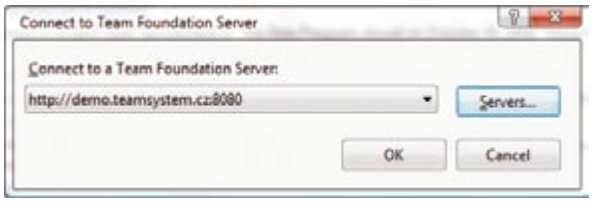
- MSF for Agile Software Development
- MSF for CMMI Process Improvement

Šablóna projektu obsahuje typy udržiavaných zoznamov, položky, ktoré sa v týchto zoznamoch nachádzajú, definície skupín používateľov a zoznam ich privilégií a oprávneni, reporty, obsah projektového portálu, politiku pre kontrolu etáp a iterácie projektu. Projektový portál obsahuje napríklad šablóny dokumentov, ktoré s projektom súvisia. Súčasťou šablóny projektu je aj integrovaná nápoveda podľa vybranej metodológie

Pre svoj projekt si môžeme zvoliť jednu z vymenovaných šablón metodológii, prípadne ak máme špecifické požiadavky, ktoré sa nedajú „zaškatuľkovať" ani do agilnej, ani do striktnej CMMI metodológie, môžeme si vytvoriť vlastnú šablónu. Pri modifikácii existujúcej šablóny, alebo vytváraní novej môžeme vychádzať z niektorej preddefinovanej metodiky, prípadne si môžeme navrhnúť metodiku vlastnú, Taktiež môžeme modifikovať, alebo vytvárať vlastné zoznamy, reporty, alebo obsah projektového portálu. Fyzicky vytvoríme nový projekt v nemu vývojového prostredia pomocou položky „New Team Project".



Menu pre vytvorenie nového tímového projektu



Pripájanie novo vytváraného projektu k Team Foundation Serveru

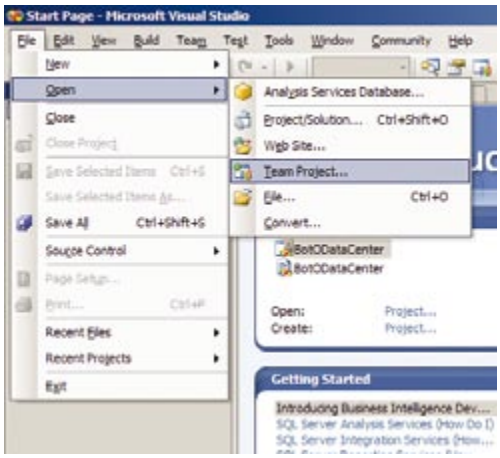
Po zadaní názvu projektu vyberieme metodologickú šablónu. Pre vytvorenie nového projektu musíme ešte zadať dva parametre – názov portálu a adresár, kde budú uložené súbory tvoriace projekt.

Otvorenie tímového projektu a pripojenie sa k Team Foundation Serveru

Pri vytváraní tímového projektu vlastne vytvárame pripojenie na Team Foundation Server. Prvým úkonom bude spustenie vývojového prostredia Visual Studio 2005. Pri štandardnej inštalácii na vývojárskom počítači spustíme vývojové prostredie z hlavného menu operačného systému

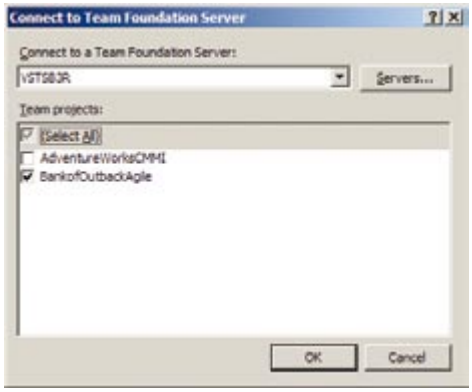
„Start | All Programs | Microsoft Visual Studio 2005 | Microsoft Visual Studio 2005"

Pre vytvorenie prepojenia na Team Foundation Server využijeme hlavné menu vývojového prostredia „File | Open | Team Project“



Založenie nového tímového projektu

Položka menu „Team Project“ aktivuje dialóg pre vytvorenie prepojenia na Team Foundation Server. V dialógu vyberieme najskôr server, na ktorý sa chceme pripojiť a následne projekt zo zoznamu tímových projektov.

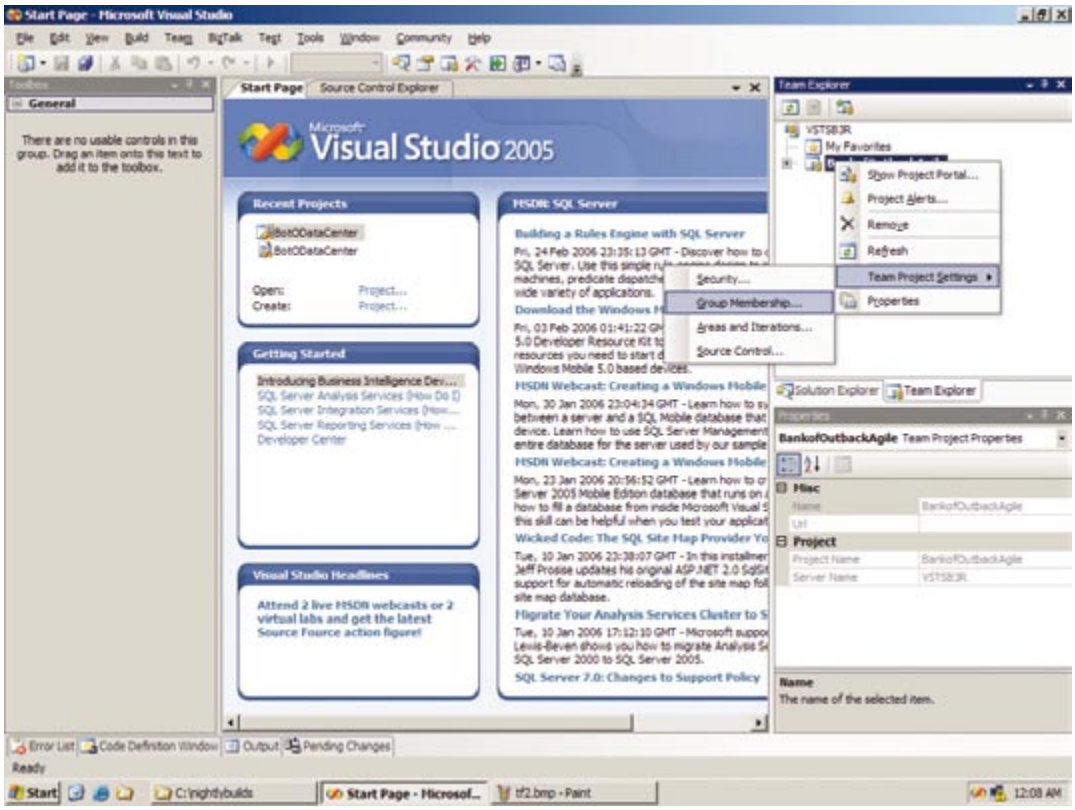


Pripojenie na Team Foundation Server

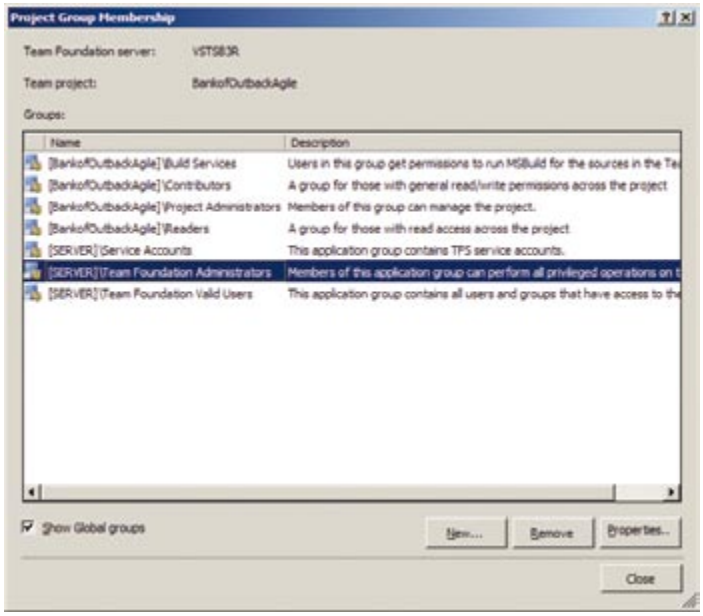
Pridanie nového používateľa

Používateľské účty a skupiny používateľov môžu byť vytvorené buď pre celý server, alebo izolovane a diferencovane pre jednotlivé prohekty tímovej spolupráce. Jednotlivým skupinám používateľov a prípadne ai používateľom jednotlivo sú potom prideľované potrebné oprávnenie, napríklad na vytvorenie buildu, zverejňovanie a posudzovanie výsledkov testovania a podobne. Jednou z najdôležitejších úloh pri kreovaní pracovného tímu je pridanie nového používateľa do skupiny používateľov Team Foundation Server Administrator Group. Skupiny používateľov je možné využívať už existujúce v Active Directory, alebo môžeme vytvárať skupiny používateľiov a používateľské účty samostatne pre Team Foundation Server. Ako východiskovú situáciu predpokladáme otvorený projekt pripojený k Team Foundation Server, teda situáciu na konci scenára v predchádzajúcom odseku. Administrátorský úkon pridania nového používateľa realizujeme v kontextovom menu tímového projektu. Kontextové menu aktivujeme kliknutím pravého tlačidla myši na názov projektu v okne Team Explorer (štandardne je situované v pravom hornom roh pracovnej obrazovky vývojového prostredia), V kontextovom mene vyberieme vnorenú položku „Team Project Settings | Group Membership“

V zozname typov pracovných skupín vyberieme skupinu „[SERVER]\Team Foundation Administrators“. Dvojklikom na názov skupiny sa zobrazí dialóg pre nastavenie parametrov pre túto pracovnú skupinu



Kontextové menu pre administráciu členov pracovných skupín

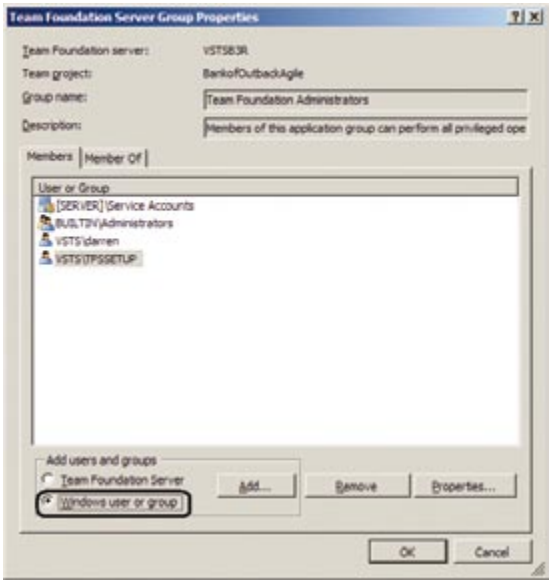


Výber zo zoznamu pracovných skupín

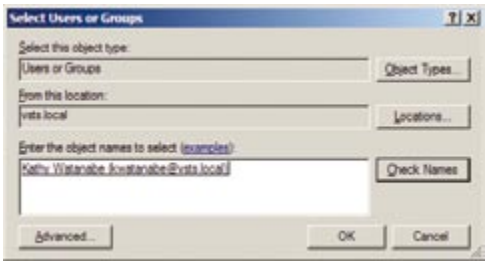
V dialógu „Team Foundation Server Group Properties" si všimnime ovládacie prvky zoskupené v sekcii (group boxe) „Add users and groups". V tejto sekcii označíme voľbu „Windows user or group"

Následne pridáme používateľa alebo skupinu tlačidlom „Add".

V dialógu „Select Users or Groups box" vyhľadáme príslušného používateľa. Pre vyhľadanie stačí napísať jeho meno, alebo prihlasovacie meno do editačného textového poľa. Potom tlačidlom „Check Names" vyhľadáme úplnú informáciu o hľadanom používateľovi. Po jeho potvrdení sa tlačidlom „OK" vrátime do dialógu „Team Foundation Server Group Policy". Rovnakým postupom, teda tlačidlom „OK" sa vrátime do dialógu „Select Group Membership"



Dialóg „Team Foundation Server Group Properties"

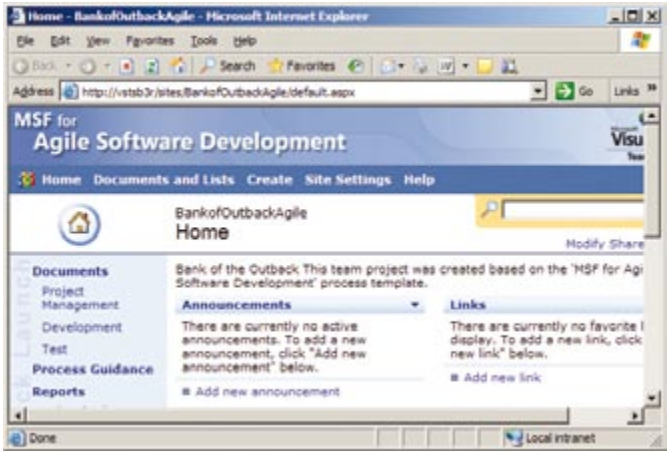


Pridelenie člena zo zoznamu používateľov alebo skupín kreovaných na úrovni operačného systému Windows

V dialógu „Team Foundation Server Group Properties" skontrolujeme, či pridanie používateľa bolo úspešné a nami pridávaný používateľ je v zozname používateľov.

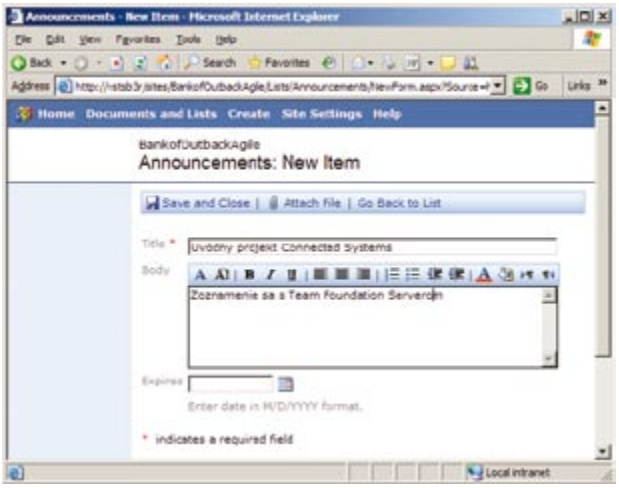
Vytvorenie nového oznámenia pre členov tímu na portáli projektov

Prácu členov tímu je potrebné nielen koordinovať, ale aj informovať o aktuálnych faktoch a udalostiach. Pre tento účel slúži na vývojárskom portáli zložka oznámení „Announcements". Ak začíname pracovať s novým tímovým projektom, bude zatiaľ zoznam oznámení prázdny Zobrazíme ju v prehliadači webového obsahu na hlavnej stránke portálu projektov „SharePoint Project Portal". Pre pridanie nového oznámenia na hlavnej stránke aktivujeme odkaz „Add new announcement". Kliknutím na tento odkaz sa zobrazí zoznam dokumentov v zložke „Announcements".



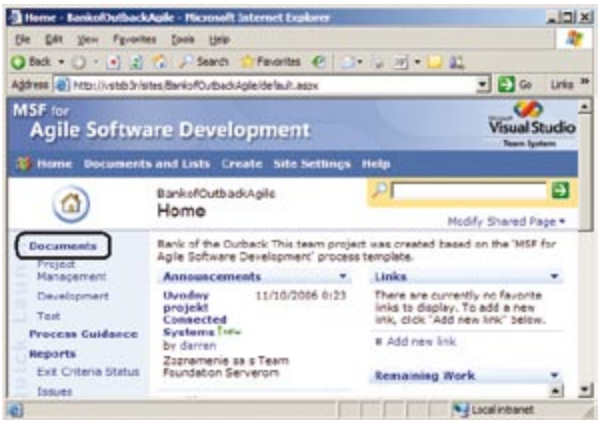
Hlavná stránka projektového portálu „SharePoint Project Portal"

Pri vytváraní nového oznámenia zadáme do poľa „Title" jeho názov (nadpis) a do poľa „Body text" vlastný text oznámenia. Formulár pre vytváranie nového oznámenia umožňuje pomocou tlačidlovej lišty známej z textového editora Word vytvárať formátovaný text.



Formulár pre vytváranie nového oznámenia

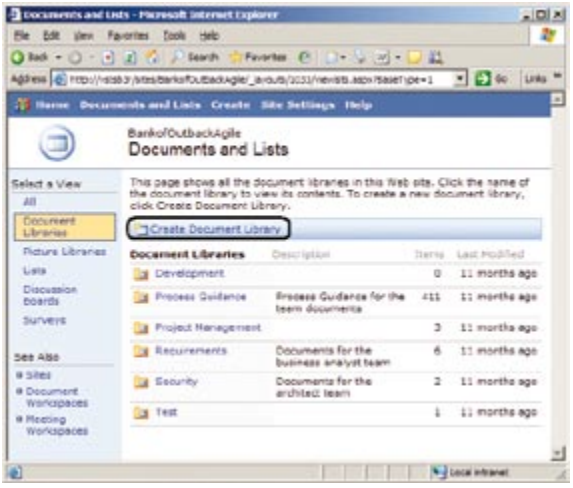
Vytváranie nového oznámenia ukončíme tlačidlom „Save and Close“. Človek je v princípe tvor nedôverčivý, takže pridanie nového oznámenia môžeme operatívne skontrolovať na hlavnej stránke projektového portálu.



Sekcia oznámení na projektovom portáli

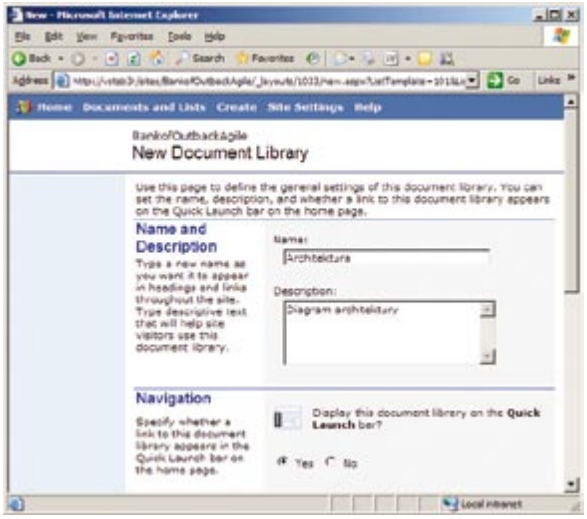
Vytvorenie knižnice dokumentov na portáli projektov

Pre prácu s knižnicou dokumentov je potrebné aktivovať link „Documents“ v ľavom zvislom paneli na pracovnej ploche stránky portálu (viď posledný obrázok z predchádzajúceho odseku). Na toolbare stránky knižnice dokumentov aktivujeme odkaz „Create Document Library“



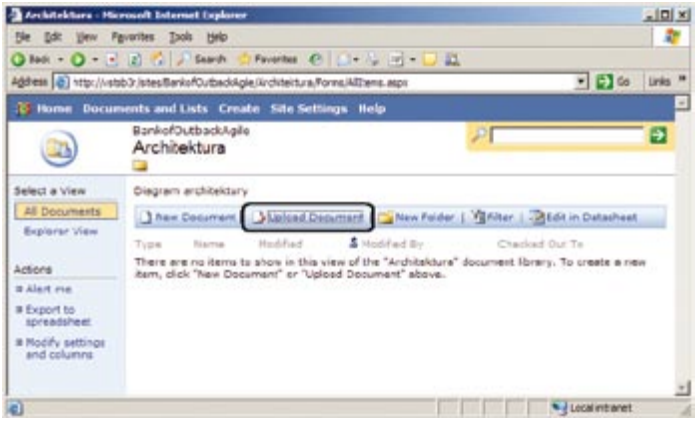
Odkaz pre vytvorenie knižnice dokumentov

Vo formulári pre vytvorenie novej knižnice dokumentov je potrebné zadať názov a keďže to bude knižnica dokumentov zdieľaná viacerými používateľmi bude vo väčšine prípadov vhodné ju v poli „Description text“ aspoň jednou dvoma vetami popísať, naprklad aké typy dokumentov bude obsahovať a podobne. Nakoniec tlačidlom „Create“ vytvorenie knižnice potvrdíme.



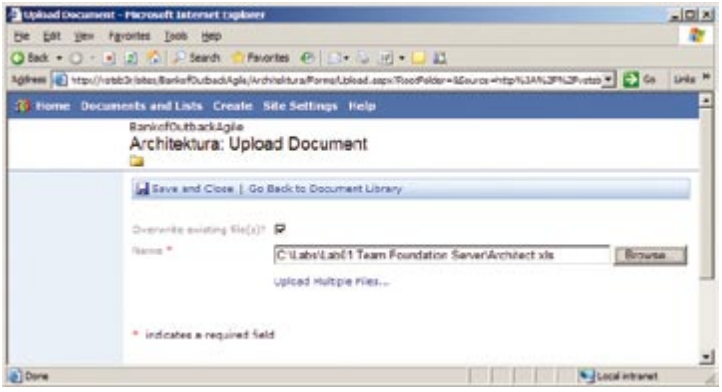
Vytvorenie novej knižnice dokumentov

následne pomocou tlačidla „Upload document“ uploadujeme dokument na server.



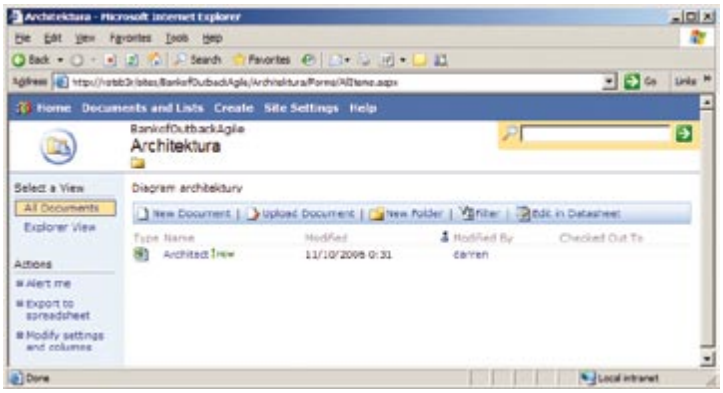
Link pre upload dokumentu na server

Napokon umiestnime do knižnice existujúci dokument. V našom prípade sme umiestnili do zložky dokumentov „Architektúra“ dokument vopred pripravený v tabuľkovom procesore Microsoft Excel. Vo formulári pre upload dokumentu je potrebné zadať úplnú cestu pre prístup k uploadovanému dokumentu, ktorý sa zatiaľ nachádza na klientskom počítači

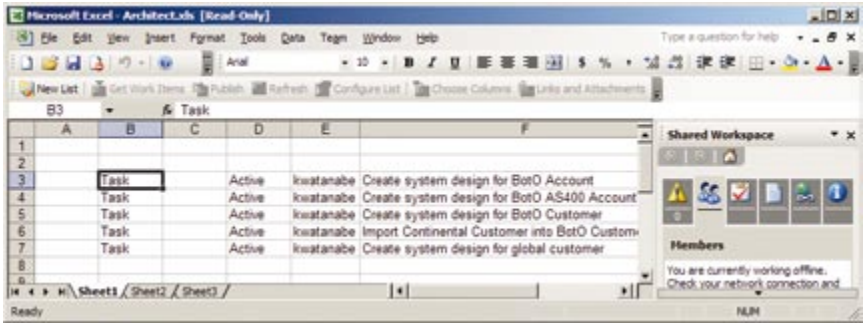


Upload dokumentu na server

Nakoniec si môžeme overiť, že uploadovaný dokument sa nachádza na serveri v zložke dokumentov „Architektúra“. Kliknutím na názov dokumentu ho môžeme na lokálnom počítači otvoriť v aplikácii v ktorej bol vytvorený. Náš dokument bude teda otvorený v tabuľkovom procesore Excel

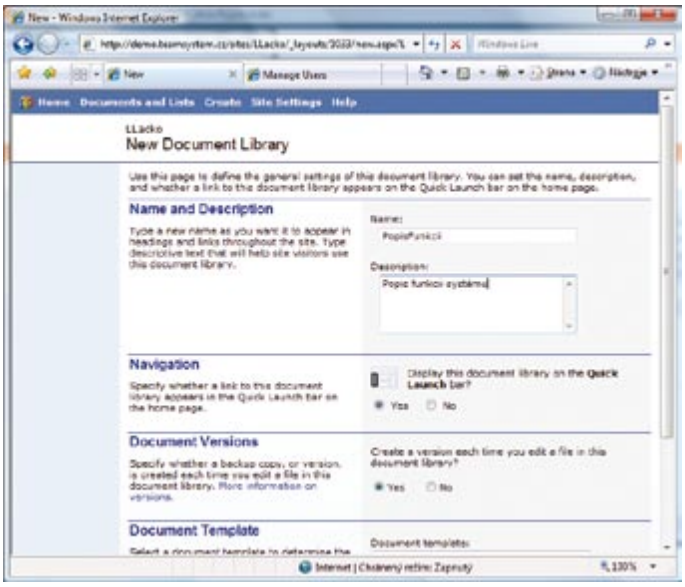


Uploadovaný dokument sa nachádza na serveri v zložke dokumentov „Architektúra“



Otvorenie dokumentu uloženého na serveri v programe Excel na lokálnom počítači

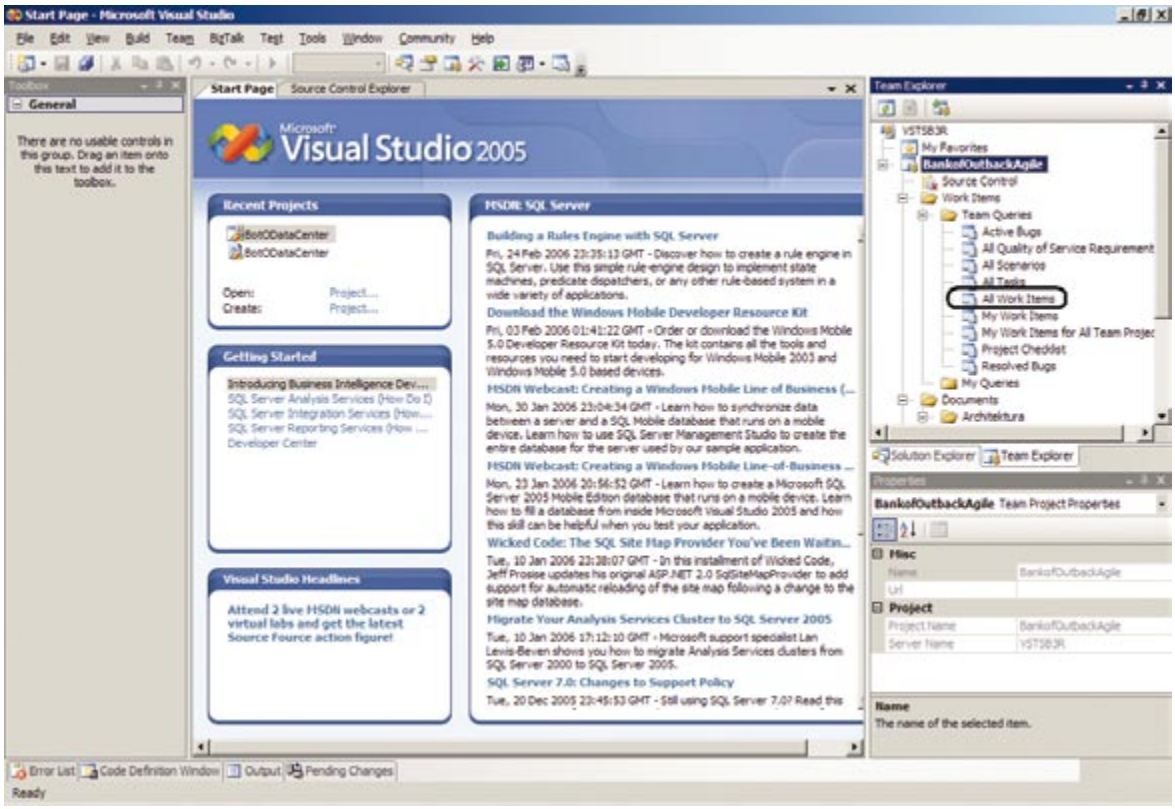
Pri vytváraní dokumentu je potrebné uvedomiť si o aký druh dokumentu sa jedná a prediktívne sa zamyslieť nad jeho životným cyklom. Ak budeme dokument často aktualizovať, napríklad v prípade ukladania analýz, je potrebné zapnúť verziovanie dokumentu. V sekcii „Document Versions“ prepneme ovládací prvok typu Radio button na hodnotu „Yes“. Pri zapnutom verzionovaní sa môžeme vrátik k niektorej z predchádzajúcich verzií apokračovať v práci. Verziovanie funguje len v tých aplikáciách pre vytváranie a editovanie dokumetov, ktoré podporujú spoluprácu so službou Microsoft Windows SharePoint Services. Takéto aplikácie sú Word 2003, Excel 2003, a PowerPoint 2003. Pozor, ak dokument z knižnice dokumentov vymažeme, budú vymazané aj všetky jeho uschované predchádzajúce verzie.



Pri vytvárani dokumentu je možné zapnúť verziovanie dokumentu

Integrácia vývojového prostredia Visual Studio 2005 Team System s portálom SharePoint, a aplikáciami Excel a Project

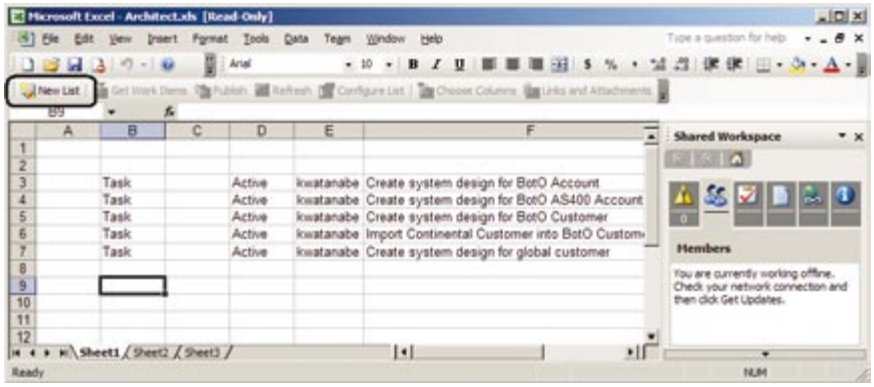
Pre demonštráciu možností integrácie vývojového prostredia Visual Studio 2005 Team System s jeho najbližším IT okolím, hlavne s aplikáciami Excel a s portálovou aplikáciou SharePoint Portal Server. Najskôr sa pokúsime o prístup k excelovskému dokumentu, ktorý sme v predchádzajúcej stati uloadovali do novovytvorenej zložky dokumentov „Architektura". V okne „Team Explorer" v pravom hornom časti pracovnej obrazovky vývojového prostredia otvoríme zložku „Documents". V nej nájdeme nami vytvorenú podzložku „Architektura" a v nej uploadovaný dokument



Zložka „Team Queries"

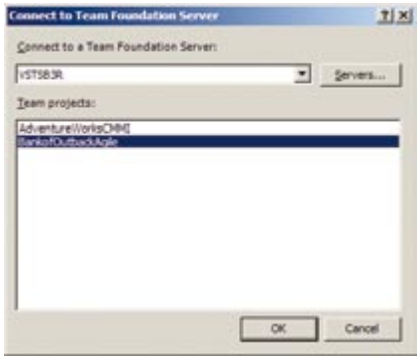
Úpravy dokumentu v programe Excel

Po otvorení dokumentu v programe Excel ukážeme spoluprácu tohto programu so serverom Team Foundation Server. V excelovskom dokumente si vyberieme vhodnú prázdnu bunku, najlepšie v druhom stĺpci zľava o riadok nižšie pod dosiaľ existujúcimi položkami.



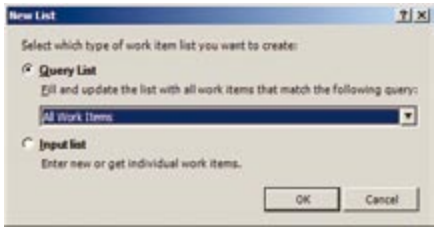
Vytvorenie nového zoznamu pripojeného na Team Foundation Server v dokumente programu Excel

Tlačidlom „New List" zobrazíme dialóg pre pripojenie sa na Team Foundation Server. Pomocou ovládacích prvkov dialógu „Connect to Team Foundation Server" vyberieme príslušný server a projekt.



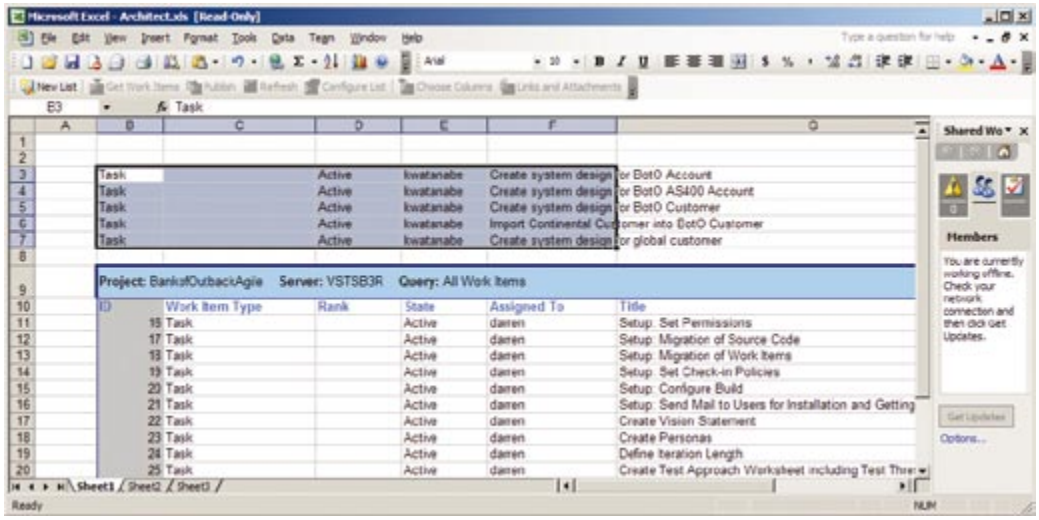
Pripojenie sa na Team Foundation Server

V dialógu „New List" aktivujeme ovládací prvok typu radio button „Query List" a pomocou prvku typu combo box pod ním vyberieme položku „All Work Items"



Pripojenie sa na Team Foundation Server

Označíme bunky, ktoré chceme skopírovať do projektu a pomocou dvojice funkcií „Edit | Copy" a „Edit | Paste" ich umiestnime na vhodné miesto dokumentu

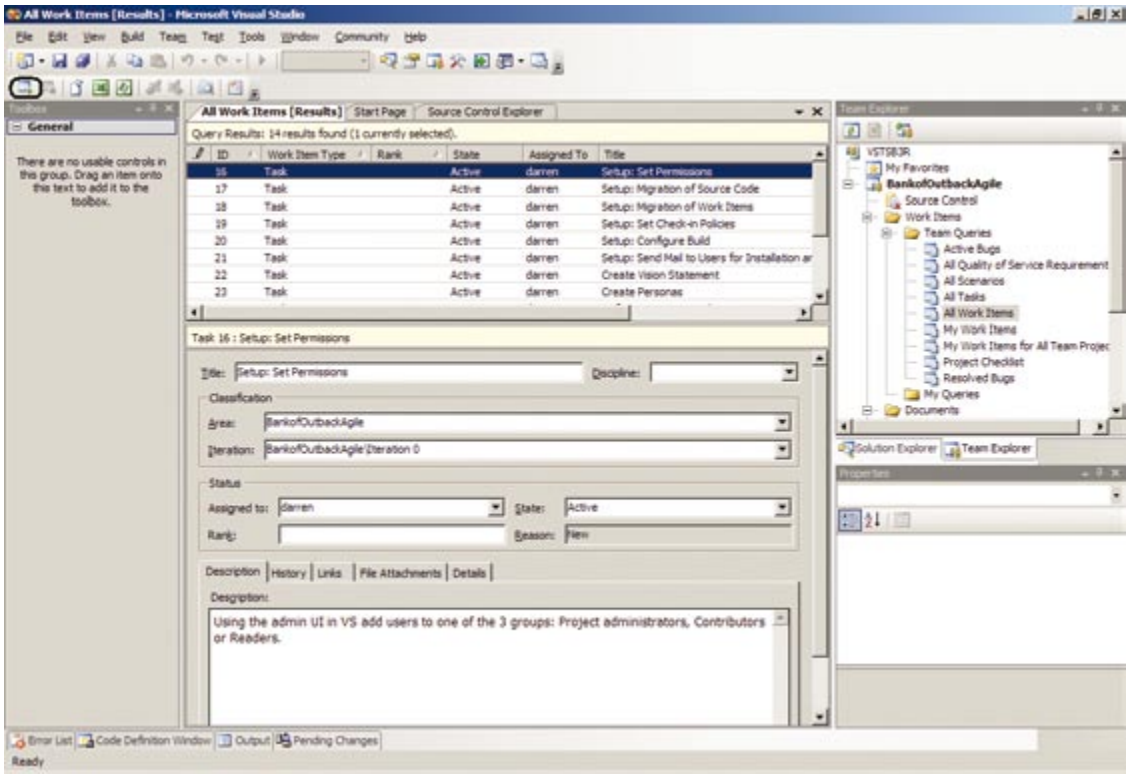


Kopírovanie položiek do zdieľaného dokumentu

Po vykonaní zamýšľaných úprav je potrebné takto upravený dokument tlačidlom „Publish" publikovať na server. Po tomto úkone môžeme program Excel zavrieť. Nakoľko zmeny už boli publikované na server ich nemusíme ukladať do súboru dokumentu, ktorý je v konečnom dôsledku aj tak len dočasný. Nakoľko sa jedná o tímovú spoluprácu, s jedným dokumentom môžu pracovať viacerí používatelia, pričom by mohlo dochádzať k rôznym konfliktom, najmä v prípade ak niektorú hodnotu v rovnakom čase menia viacerí používatelia. Možnosť takéhoto konfliktu je ošetrená na úrovni editovania a zápisu polí. Zmeny vykonané jedným používateľom sú pre ostatných viditeľné až po ukončení editovania a publikovaní dokumentu

Zobrazenie úprav vo vývojovom prostredí

Vo vývojovom prostredí sa vrátime sa do okna pracovnej obrazovky „Team Explorer“. Expandujeme zložku „Work Items | Team Queries“ a vyberieme položku „All Work Items“. Všimnite si, že do okna v strede pracovnej plochy obrazovky pribudla záložka „All Work Items [Results]“. Keď bude táto záložka aktívna (teda na povrchu stredného okna pracovnej plochy), môžeme sa dotazovať na hodnoty dokumentu tlačidlom „Run Query“



Okno vývojového prostredia, výber zložky „All Work Items“. Tlačidlo „New Query“ je označené rámikom

Pre aktívnu položku v okne „All Work Items [Results]“ je možné zobraziť podrobnosti v dolnom okne, ktoré je rozdelené do záložiek

- Description
- History
- Links
- File Attachments
- Details

Spolupráca s aplikáciou MicrosoftProject

Úzkym miestom pre vývoj softvérových produktov je čas. Ak vývojárske práce do ktorých je spravidla zapojení kolektív kvalifikovaných odborníkov trvajú príliš dlho, negatívne sa to podpíše nielen na nákladoch, ale aj na konkurencieschopnosti. História si totiž pamätá len prvých a najlepších, takže keď prídeme s produktom nie príliš excelentnej kvality niekoľko mesiacov po konkurencii, ani najbrilantnejší marketing s tým veľa nezmôže. Ale aj v prípade, že si rozdelíme čas medzi jednotlivé projekty tak, že na každý z nich budeme mať dostatok času, neznamená to automaticky, že toto množstvo času, ktoré sme pre daný projekt ochotní obetovať aj účelne využijeme. V tomto úsilí nám môže výrazne pomôcť produkt Microsoft Project.

Microsoft Project je primárne určený pre riadenie času a zdrojov, multiprojektové plánovanie, plánovanie a riadenie nákladov, zdrojový tracking a projektové analýzy. Produkt je teda vo firme použiteľný na rozličných úrovniach, od operatívneho plánovania až po vrcholový manažment,.

V prostredí MS Project môžeme názorne získať prehľad o jednotlivých etapách napríklad vo forme grafického znázornenia na časovej osi (Ganttov diagram). Na tomto diagrame vidia všetci členovia tímu

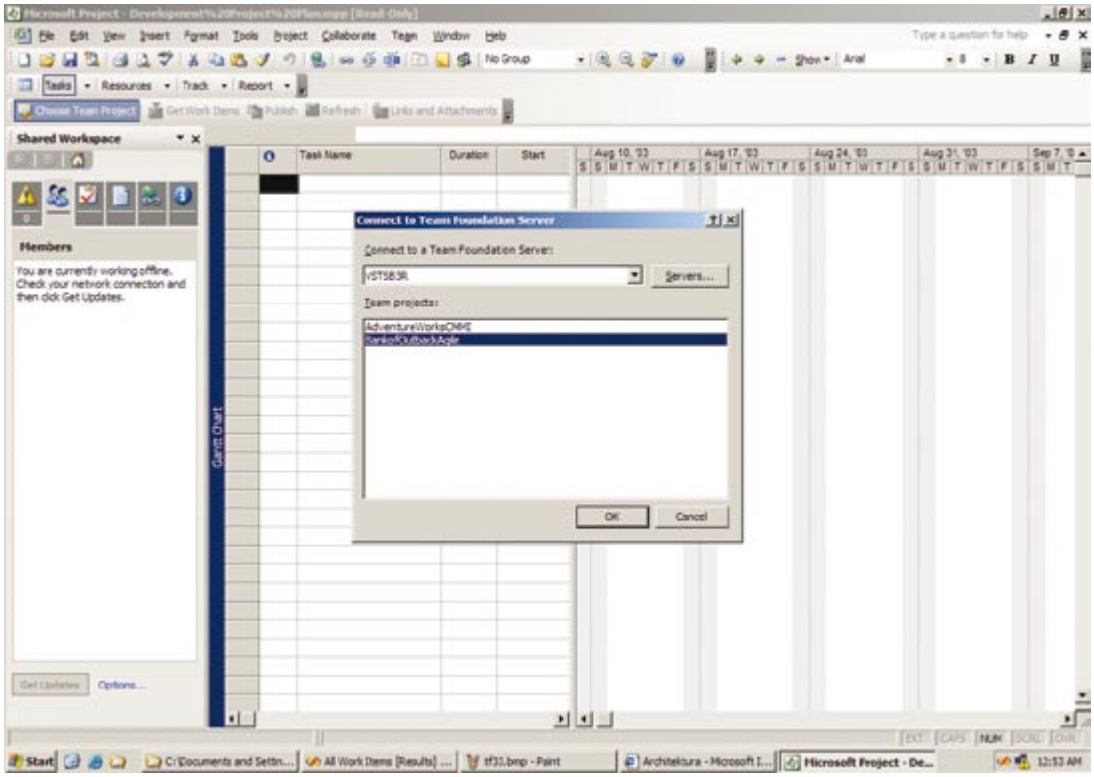
svoje vlastné úlohy a ciele zaznamenané graficky. Zobrazenie je možné jednoducho upraviť a prispôsobiť napríklad presúvaním stĺpcov, filtrovaním podľa úloh, zoskupovaním podľa projektov a podobne. Je možné zobraziť položky zoznamu úloh v aplikácii Microsoft Outlook. členovia tímu tak budú mať všetky svoje úlohy sústredené na jednom mieste.

Ak sa pozrieme na návrhy šablóny pre vývoj softvéru, trochu úsmevne môžeme konštatovať, že v Microsofte sa poučili a etapu testovania softvéru navrhli oveľa dlhšiu ako etapu jeho vývoja. A na spoľahlivosti ich produktov to v poslednej dobe aj vidno.

Nie vždy nám bude vyhovovať niektorá zo šablón. Ak navrhujeme harmonogram nejakého menej typického projektu, je výhodné si harmonogram navrhnúť sám. Celý postup je prehľadne rozložený do niekoľkých krokov. Najskôr je potrebné určiť dátum začiatku projektu, stanoviť pracovnú dobu vo firme a zadať, či náš projekt bude spolupracovať s **Microsoft Project Serverom** a portálom **Microsoft Project Web Access**. Microsoft Project Server CAL umožňuje použitie webového portálu Microsoft Project Web Access. Klienti potom pre prístup k informáciám potrebujú iba Microsoft Project Server CAL a navzájom spolupracujú prostredníctvom webového prehliadača. Potom navrhujeme jednotlivé etapy a fázy projektu, pričom samozrejme musíme zohľadniť, že niektoré fázy a etapy môžu byť zahájené až po ukončení etáp na ktoré naväzujú.

Tímová spolupráca má samozrejme veľa výhod. Ak niektorý člen tímu zistí, že do projektu nebola niektorá dôležitá úloha zahrnutá, môže ju do projektu dodatočne pridať. Takýmto spôsobom vlastne všetci členovia tímu pomáhajú pri návrhoch a úpravách projektu. Samozrejme nové úlohy budú „naostro" zaradené do projektu až po ich schválení vedúcim projektu. Taktiež člen tímu (ak má na to poverenie) môže časť svojich úloh delegovať na iných členov tímu.

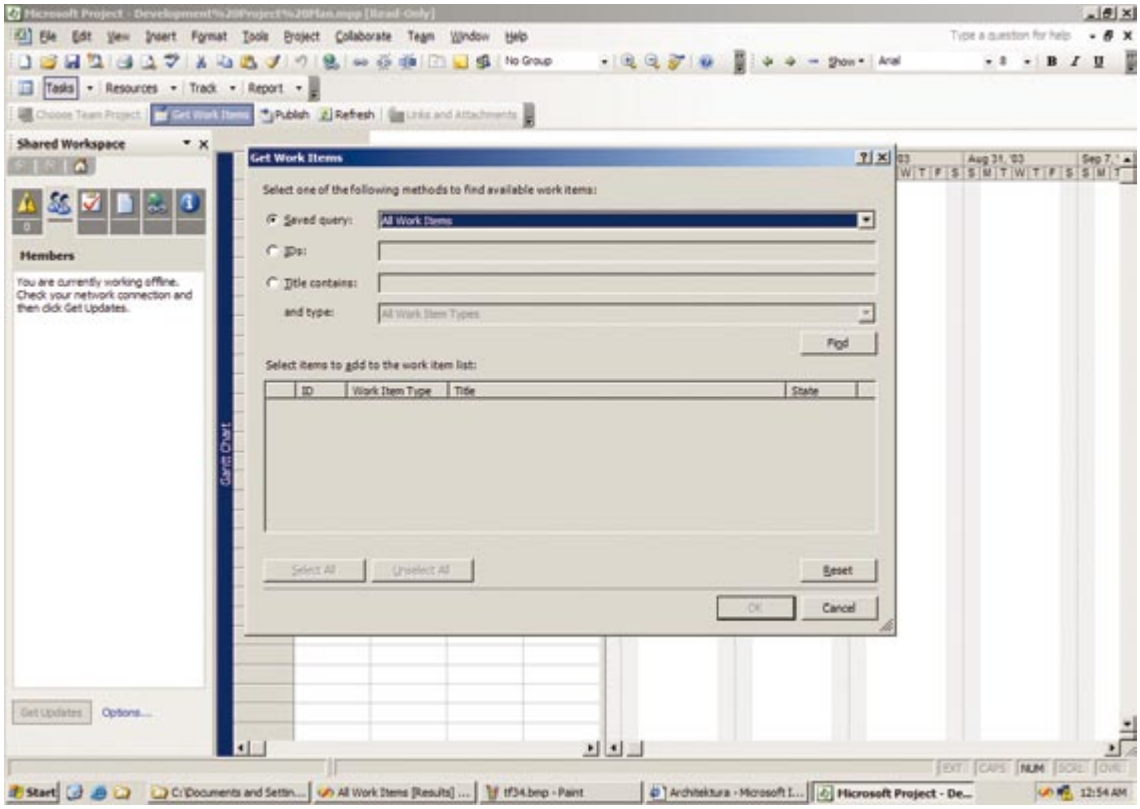
K projektovému plánu v programe Microsoft Project sa dostaneme z vývojového prostredia Visual Studio 2005 Team System. V okne „Team Explorer" expandujeme zložku „Project Management". Dvojklikom na názov vývojového projektu „Development Project Plan.mpp" otvoríme projektový plán v aplikácii Microsoft Project.



Výber tímového projektu v programe Microsoft Project

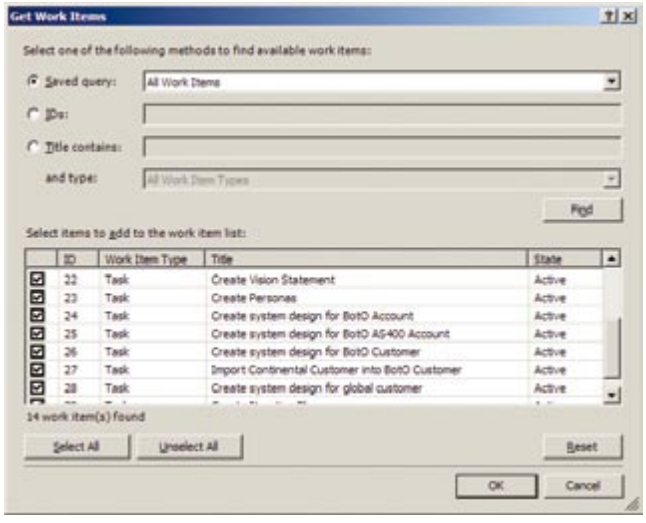
Pre výber tímového projektu v programe Microsoft Project aktivujeme tlačidlo „Choose Team Project". Zobrazí sa dialóg pre pripojenie sa na Team Foundation Server. Pomocou ovládacích prvkov dialógu „Connect to Team Foundation Server" vyberieme príslušný server a projekt.

Tlačidlom „Get Work Items“ otvoríme rovnomenný dialóg. V ňom môžeme ponechať označený ovládací prvok radio – button „Saved query“ . Pomocou vľavo situovaného prvku typu combo – box vyberieme voľbu „All Work Items“ a tlačidlom „Find“ spustíme vyhľadávanie



Vyhľadávanie projektov

Následne sa zobrazí zoznam projektov, ktoré vyhovujú vyhľadávacím kritériám.



Zoznam projektov, ktoré vyhovujú vyhľadávacím kritériám

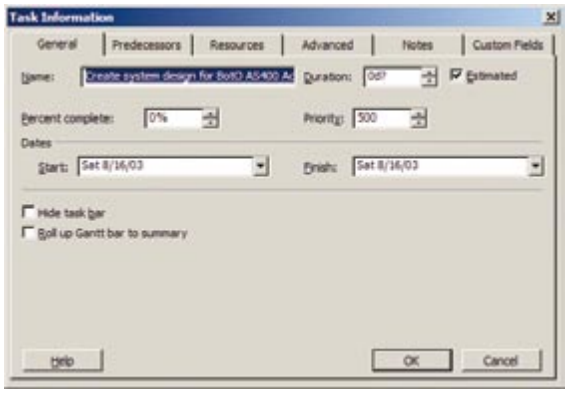
O každom projekte si môžeme nechať zobraziť detailné informácie. Dialóg pre zobrazenie informácií je prehľadne rozdelený do záložiek

- General
- Predcessors
- Resources
- Advanced

– Notes
– Custom Field

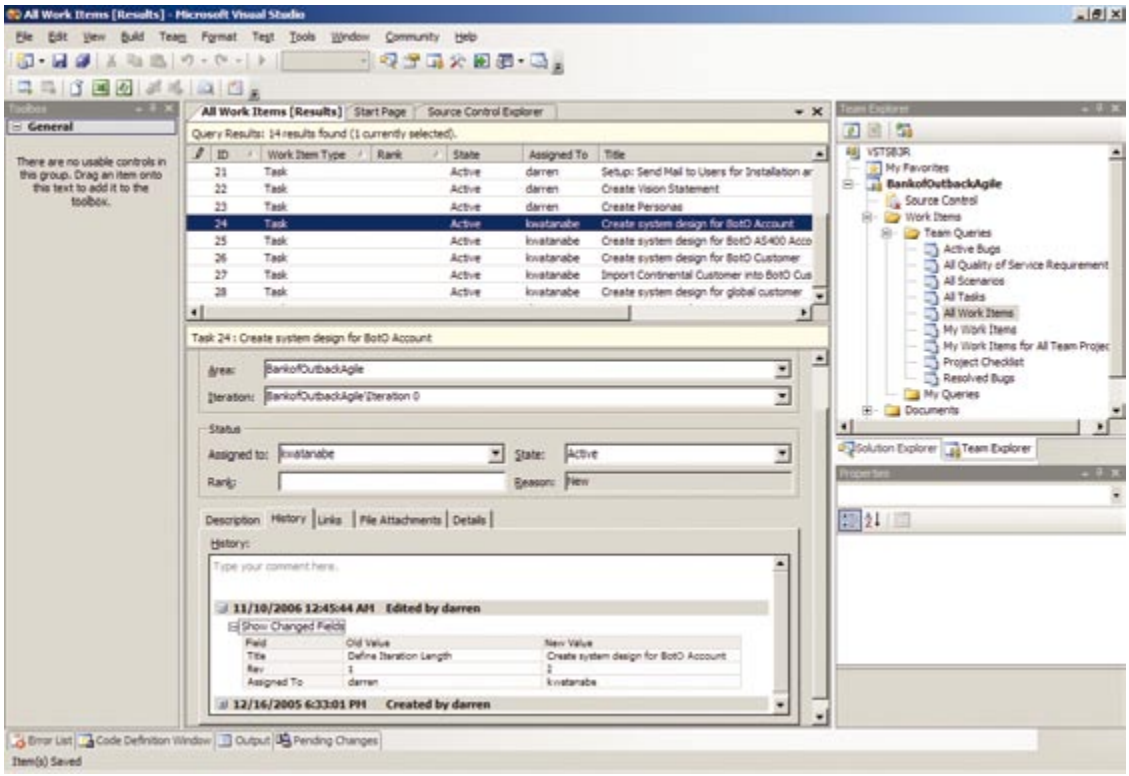
V záložke „Resources" môžeme nechať zobraziť na ktorých častiach projektu a ako participujú jednotliví členova tímu.

Samozrejme môžeme plánovanie projektu meniť. Napríklad v záložke „General" môžeme operatívne zmeniť trvanie niektorej etapy projektu a podobne.



Detailné informácie o projekte

Históriu zmien vykonaných v projekte v aplikácii MS Project môžeme kedykoľvek zobraziť vo Visual Studiu 2005



História zmien vykonaných v aplikácii MS Project zobrazené vo Visual Studiu 2005

Je potrebné si uvedomiť, že pri publikovaní sa na server pošle len podmnožina údajov – riadky s nastavením príznaku pre publikovaní. Tento príznak objavíte pri skrolovaní riadkov doprava. Preto je potrebné súbory ukladať, spravovať a udržiavať mimo projektový server. Taktiež je potrebné uchovávaťaj súbory .MPP, nakoľko niektoré dôležité záležitosti, ako napríklad ceny zdrojov alebo nadväznosť úloh sa v tejto verzii s projektovým serverom nesynchronizujú.

KAPITOLA 2: NÁVRH DISTRIBUOVANÝCH SYSTÉMOV

Zdalo by sa, že vývoj a nasadenie distribuovaných systémov odstráni veľa problémov v oblasti podnikových aplikácií, no ako pre všetky oblasti IT, aj pre tieto systémy platia Murphyho zákony. Podľa nich odstránenie jedného problému prinesie dva ďalšie. U distribuovaných aplikácií vznikajú nedorozumenia hlavne medzi vývojármi na jednej strane a administrátormi nimi vyvyutých aplikácií na strane druhej. Vývojári nemajú dostatok poznatkov a skúseností s prevádzkovým prostredím. Administrátori sa zasa dostanú k aplikáciám buď na konci etapy testovania, alebo až po jej ukončení, takže často nerozumejú jej koncepcii, filozofii a nezriedka ani jej nárokom a požiadavkám.

Tieto problémy môže pomôcť odstrániť etapa modelovania v procese návrhu architektúry a koncepcie systému. Podstatou činnosti každého informačného systému, ktorý využíva databázy je transformácia informácií z vonkajšieho sveta na dáta. Tento proces môžeme na rôznych úrovniach modelovať. Model informačného systému by mal byť

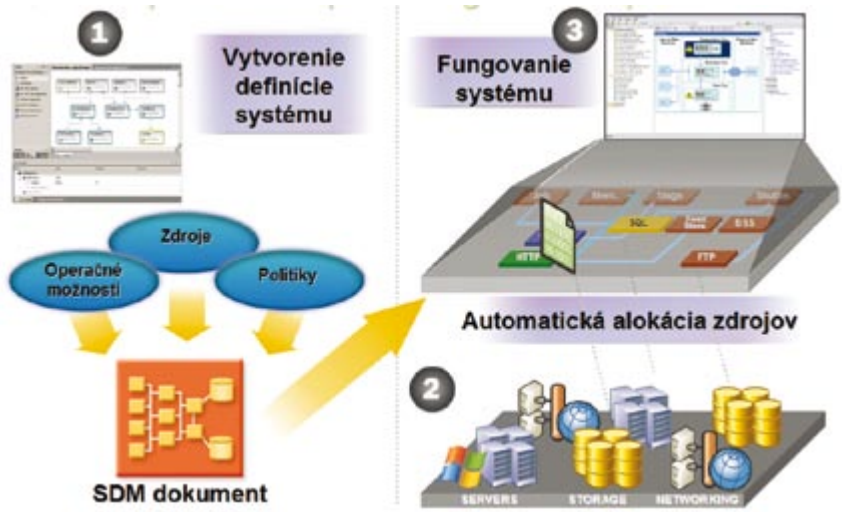
- **zrozumiteľný** – mal by vyjadrovať fakty a pravidlá v jednoduchom jazyku
- **vhodný** – mal by podchytiť čo najviac pravidiel vyplývajúcich z obchodnej logiky navrhovanej aplikácie
- **spoľahlivý** – mal by umožniť overiť si pravidlá v prirodzenom jazyku na jednoduchých príkladoch
- **stály** – je potrebné minimalizovať dosah zmien
- **vykonateľný** – model musí byť jednak realizovateľný na technickej úrovni pomocou dostupných hardvérových a softvérových prostriedkov a taktiež musí byť vhodný pre prevádzku

Modelovanie softvérovej aplikácie pozostáva z dvoch etáp
- modelovanie obchodných požiadaviek, ktoré môže byť vonkajšie (analýza vonkajších vzťahov) a konceptuálne (analýza obchodnej logiky).
- modelovanie štruktúry údajov, ktoré môže byť na logickej a fyzickej úrovni.

SDM (System Definition Model)

Procesu systémovej analýzy a systémového návrhu podľa niektorých metodík hovoríme aj konceptuálny model. Takýto model obsahuje formálny popis systému, napríklad u databázovej aplikácie popisuje údaje v databáze úplne nezávisle od ich fyzického uloženia a pri jeho návrhu sa zameriame na aplikačnú logiku ale z pohľadu človeka, nie z pohľadu neskôr použitých hardvérových a softvérových technológií. Takýto proces sa nazýva konceptuálne modelovanie. Pri tvorbe konceptuálnych modelov spravidla vnímame objekty reálneho sveta, vzťahy medzi nimi a funkcie, pomocou ktorých sa tieto vzťahy realizujú, takže konceptuálne modelovanie je v podstate objektovo orientovaný proces. Okrem objektov spravidla vstupuje do hry aj ich hierarchické usporiadanie, napríklad dedičnosť, to znamená, že objekty môžu byť vytvorené na základe iných objektov, pričom zdedia časť vlastností a podobne.

Na schéme vidíme tri fázy SDM modelovania
- vytvorenie definície systému
- automatická alokácia zdrojov
- funkcionalita systému v jej životnom cykle



Fázy modelovania s využitím koncepcie „System Definition Model"

Okrem informácií o funkcionalite obsahuje System Definition Model (SDM) aj informácie potrebné pre inštaláciu a nasadenie celého systému a to aj na úrovni jeho komponentov. Fyzicky je SDM vo VS 2005 realizovaný ako XML dokument. Obsahuje:

- zámery vývojárov aj administrátorov
- topológiu systémov
- požiadavky vývojárov
- definície politík v celom IT prostredí v ktorom bude aplikácia nasadená
- pokyny pre inštaláciu
- pravidlá pre monitorovanie a údržbu aplikácií

Visual Studio 2005 obsahuje podporu pre najvrchnejšie úrovne modelovania, pre návrh aplikácie a jej hostovania. V procese modelovania môžeme využiť

- Application Designer
- System Designer
- Deployment Designer
- Logical Datacenter Designer

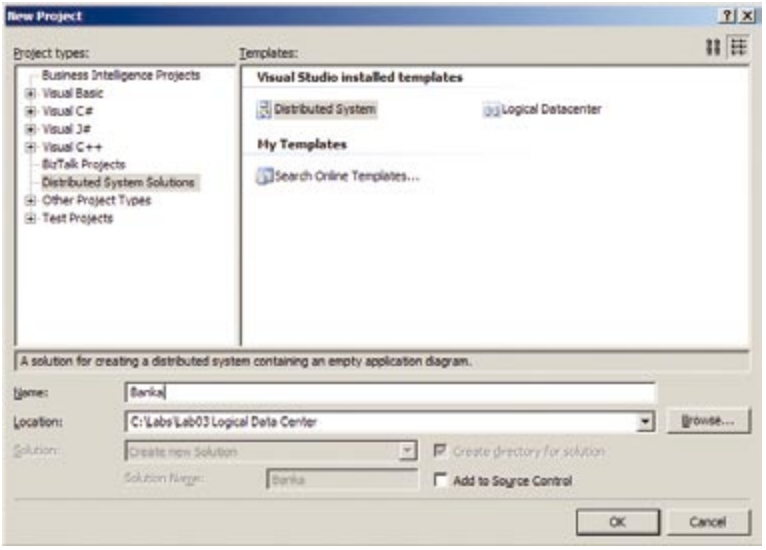
Aplikácia

Základnou jednotkou pre vytváranie modelov distribuovaných aplikácií je aplikácia. Je to v princípe jedna nedeliteľná jednotka pre nasadenie v IT prostredí. Aplikácia môže byť rôzneho typu, napríklad klasická Windows aplikácia typuWinForms, ASP.NET aplikácia, ktorá beží na serveri, webová služba a podobne. Aplikácia je verzovaná a je možné definovať rozhrania jej služieb. Tieto rozhrania sú realizované pomocou definovaných bodov – „endpointov". Tieto rozhrania sú obojsmerné – to znamená že každá jednotka (aplikácia) ponúka alebo používa služby ostatných aplikácií. Pre aplikáciu môžeme nastaviť jej konfiguračné parametre a prípadné obmedzenie na určité hostovacie prostredie. Informácie o každej základnej jednotke, teda aplikácii je uložená ako SDM dokument, teda fyzicky vo formáte XML dokumentu.

Systém

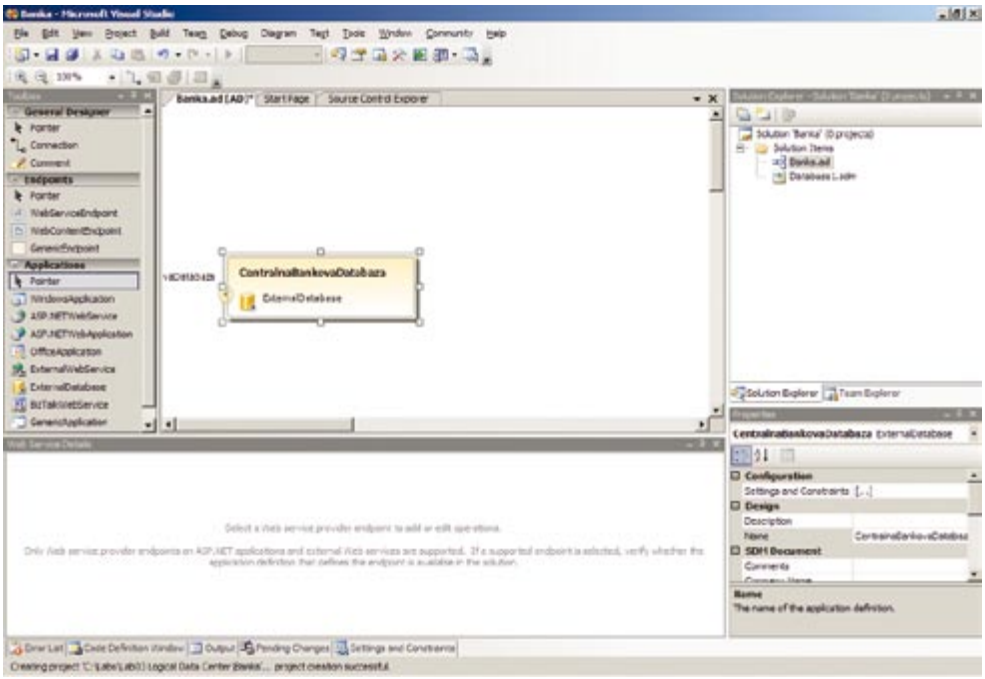
Úroveň aplikácie môžeme považovať za najnižšiu úroveň granularity. Jednotlivé aplikácie (klasické desktopové windows aplikácie, webové ASP.NET aplikácie, externé databázy a webové služby) môžeme poskladať do určitých blokov a pre tieto bloky definovať pripojenia a rozhrania jednak vo vnútri bloku medzi jednotlivými aplikáciami a taktiež aj na vyššej úrovni granularity – na úrovni blokov. Tieto bloky zložené z aplikácií nazývame system. Systtém, podobne ako aplikácia ponúka a využíva služby ostatných systémov alebo samostatných aplikácií. Na úrovni systému môžeme definovať konfiguráciu a obmedzenia aplikácií v nej uložených. Aj blok System je podobne ako blok Aplikácia uložený ako SDM dokument.

Projekt typu „Distributed System“ vytvoríme v menu „File | New | Project“ vývojového prostredia Visual Studio 2005. Vyberieme zložku projektov „Distributed System Solution“. Všimnite si, že v ponuke je okrem šablóny pre vývoj distribuovaných systémov aj šablóna pre vývoj projektov typu „Logical Datacenter“. Pri výbere tohoto typu projektu sa aktivuje Logical Datacenter Designer



Vytvorenie nového projektu typu Distributed System

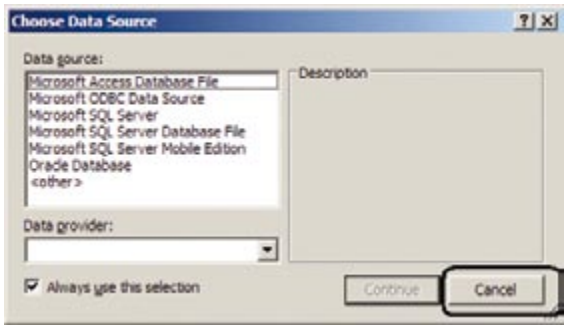
Pre návrh diagramov distribuovaných systémov je pracovná plocha vývojového prostredia rozvrhnutá rovnako ako pre iné typy vizuálnych projektov, teda v strede je okno pre vytváranie diagramu, vľavo Toolbox, odkiaľ presúvame grafické symboly do okna diagramu a vpravo okno pre zobrazenie vlastností aktuálne vybratého symbolu v diagrame. V spodnej časti je okno pre výpis procesných oznamov a hlásení, napríklad o priebehu zoatavovania projektu, popis chýb a podobne



Konfigurácia pracovnej obrazovky vývojového prostredia pre vývoj distribuovaných systémov . Symboly sa presúvajú z toolboxu na návrhovú plochu diagramu

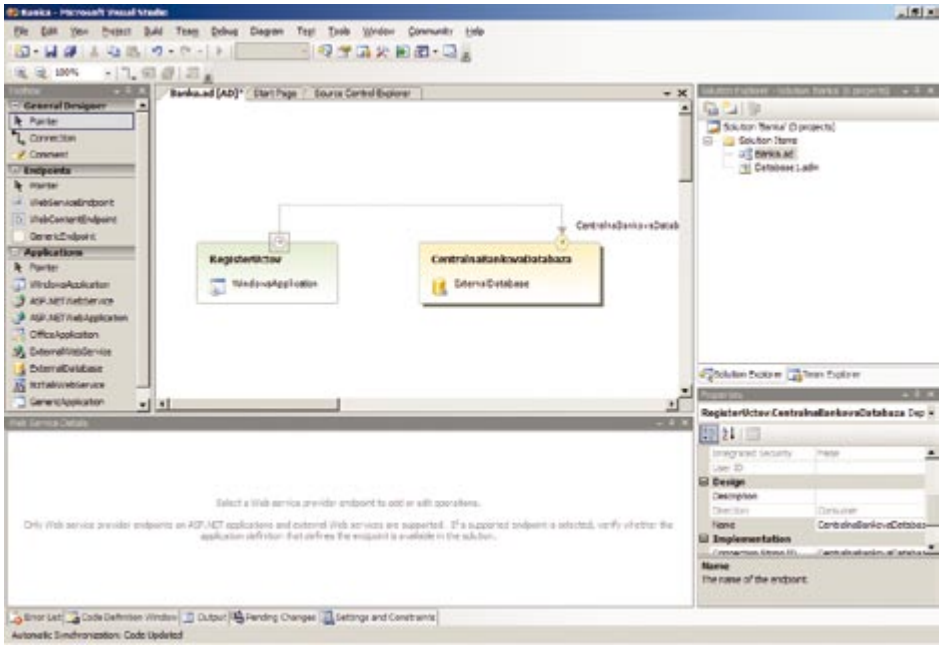
Toolbox so symbolmi pre aplikačné diagramy

Pre niektoré symboly, napríklad pre externú databázu je potrebné nastaviť parametre pre pripojenie sa k zdroju údajov. Na základe týchto údajov budú vygenerované pripojovacie reťazce v konfiguračných súboroch aplikácií, ktoré budú vygenerované na základe návrhu.



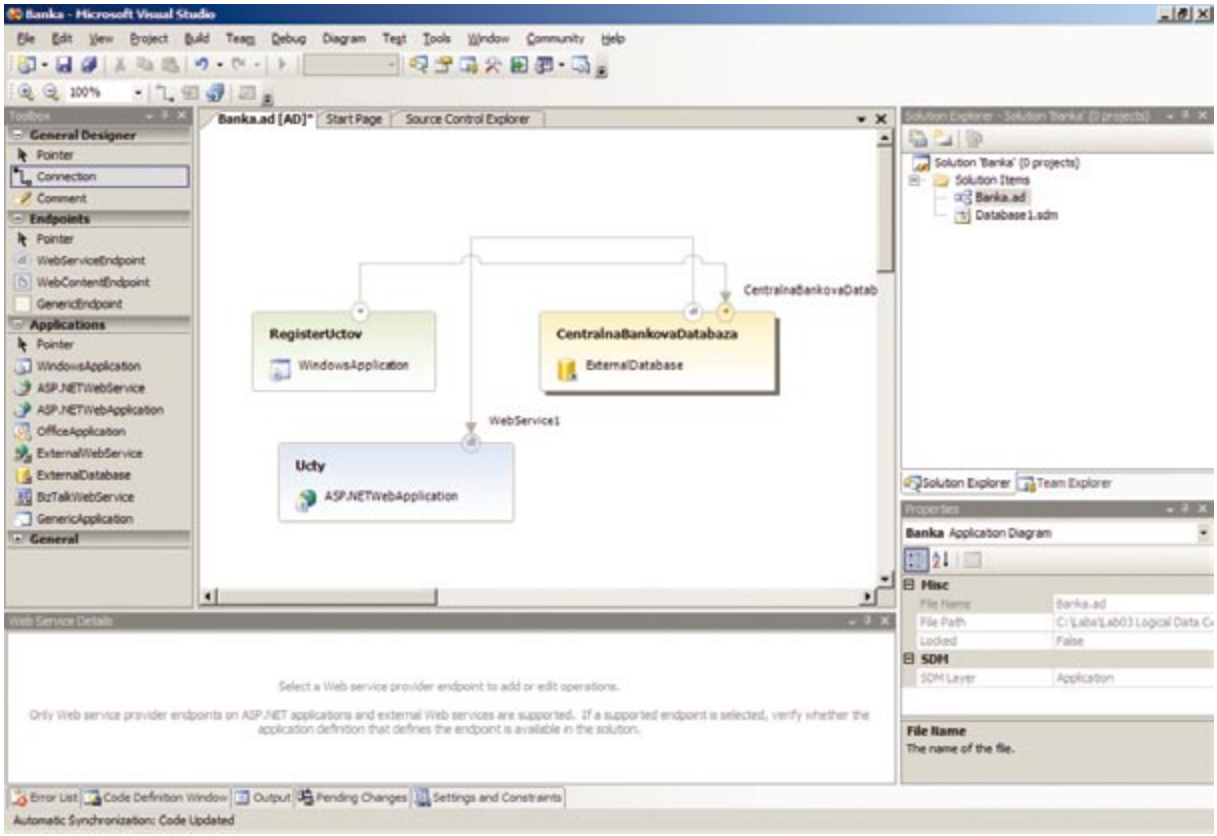
Výber dátového zdroja

Bloky sú navzájom poprepájané v zmysle aplikačnej logiky čiarami, pričom čiara je k bloku pripojená v takzvanom „end – pointe". Jedná sa vlastne o grafické znázornenie pripojenia sa ku službám poskytovaným aplikáciou, napríklad takýmto end-pointom môže byť SOAP rozhranie webovej služby.



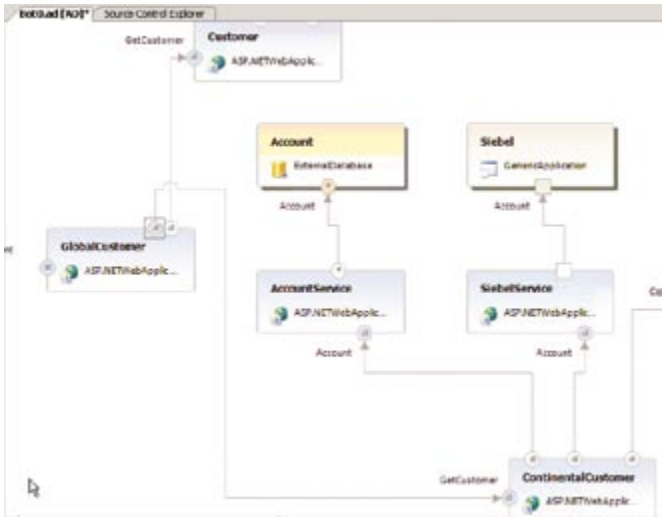
Prepojenie „end – pointov" symbolov aplikácií a databáz

Na obrázku je príklad návrhu časti systému pre bankové operácie zloženého z viacerých aplikácií. Jednotlivé obdĺžniky symbolizujú webové ASP.NET aplikácie a externú databázu. Implementovaním každého modulu vznikne vo vývojovom prostredí projekt príslušnej aplikácie. Symboly sa na plochu diagramu presúvajú z toolboxu.



Aplikačný diagram

Diagramy pre reálne systémy pracujúce v podnikovom intranetovom sú samozrejme podstatne zložitejšie ako náš príklad jednoduchého diagramu



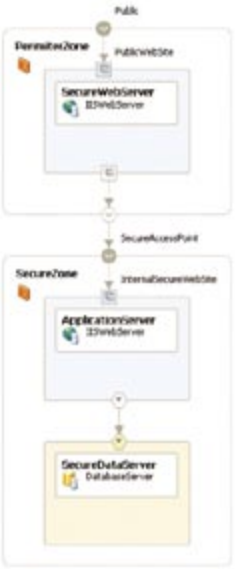
Príklad zložitejšieho aplikačného diagramu

Logické datové centrum

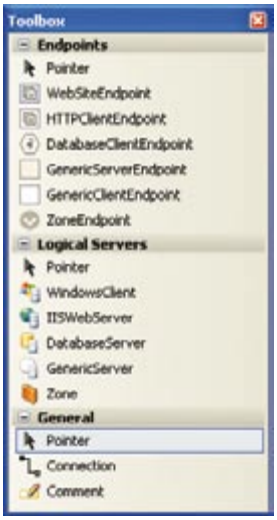
Ak si pozrieme plán architektúry databázových serverov pre väčšiu firmu, ktorá potrebuje pracovať s veľkými objemami údajov, ako sú napríklad banky, telekomunikačné spoločnosti... zistíme že infraštruktúra databázových serverov, ich vzájomné prepojenie a prepojenie s ostatnými servermi a ďalšími štruktúrami podnikovej informatiky je pomerne zložitá. Pre potreby administrátorov slúžia podrobné schémy s IP adresami prepojeniami a ďalšími náležitosťami, ktoré sú potrebné pre správu takéhoto dátového centra. Ak však takúto schému ukážeme tímu vývojárov databázových aplikácií, bude pre ich potreby zbytočne zložitá, neprehľadná, pretože obsahuje údaje, ktoré sú síce nevyhnutné pre administráciu, no pre vývoj aplikácií nemajú žiadny význam. Preto je výhodné ako podklad pre vývoj databázových aplikácií použiť logickú schému dátového centra.

Schéma logického dátového centra popisuje nastavenie a konfiguráciu serverov, komunikačné protokoly a požiadavky na autentifikáciu. Takáto zjednodušená schéma, na rozdiel od schémy pre administrátorov nepopisuje počty serverov, ani ich fyzické charakteristiky ani IP konfiguráciu, firewally, architektúru VLAN, switche, routery, kryptografiu na úrovni sieťových paketov... Nič z toho totiž pre vývoj databázovej aplikácie nie je potrebné.

Logické servery sú na schéme dátového centra rozmiestnené v logických zónach. Rozmiestnenie serverov do zón je určené napríklad vrstvou architektúry, prípadne úrovňami zabezpečenia. Medzi servermi a zónami sú definované prepojenia. Diagram logického dátového centra je uložený ako SDM.



Jednoduchá schéma logického dátového centra



Toolbox so symbolmi pre budovanie logického dátového centra

Diagram logického dátového centra môže obsahovať logické symboly týchto druhov logických serverov a aplikácií

Windows Client – hrubý klient – .NET Windows aplikácia
IISWebServer – Internet Information Server (IIS) pod správou ktorého bežia ASP.NET webové aplikácie alebo webové služby.
DatabaseServer – databázový SQL server
GenericServer – iný typ serverovej aplikácie, napríklad Microsoft BizTalk, COM+, MSMQ, Exchange...

KAPITOLA 3: DIAGRAMY TRIED

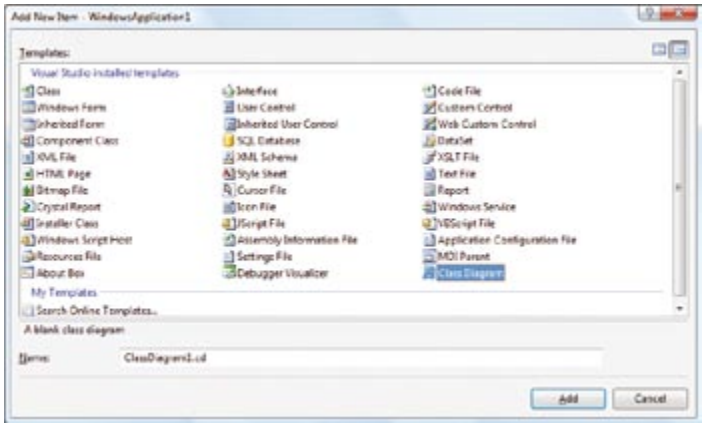
Úlohou tímu vývojárov je spravidla vytvorenie transformácie objektívnej reality z reálneho tzv. priestoru problémov do modelu. Pri návrhu a modelovaní je preto potrebné pozorovať objektívnu realitu a vytvárať jej viac alebo menej realistický obraz. Elementárnou úlohov v procese objektovo orientovaného návrhu je pomenovanie každého prvku v reálnom priestore. Na túto úlohu nadväzuje úloha vyčleniť vlastnosti prvku a určiť jeho fungovanie a správanie.

V reálnom priestore problémov najskôr identifikujeme prvok, ktorý je dôležitý pre naše zámery a ktorý sme najskôr pomenovali. Takýmto prvkom môže byť napríklad žiarovka. Prečo sme vybrali práve takýto bežný a z hľadiska programovania zdanlivo nevýznamný prvok? V tomto prípade namáme na mysli žiarovku ako prvok pre osvetľovanie, ale prvok signalizačný. Ak si pozriete obrázky z „velínov" elekrtární a technologických prevádzok z minulého storočia, žiaroviek ako symbolov prvokov a procesov tam nájdete neúrekom. Dnešné moderné riadiace systémy využívajú namiesto žiariviek spravidla grafické symboly na obrazovke.

Vráťme sa však k našej (signalizačnej) žiarovke. Prvok teda najskôr pomenujeme – dali sme mu meno „Žiarovka". Z vlastností, ktoré môže mať skutočná žiarovka v reálnom priestore vyčleníme tie, ktoré sú dôležité pre riešenie postavenej úlohy. Môže zaujímať konštrukčné riešenie žiarovky, jej vzhľad, farba, princíp fungovania, výkon, spotreba, cena... Zo všetkých týchto vlastností vyberieme iba niektoré.
V našom prípade nás bude zaujímať, či žiarovka v danom okamihu svieti alebo nie. Okrem tejto základnej vlastnosti, môžeme vyšpecifikovať prípadne aj ďalšie vlastnosti. Poslednou úlohou je určiť správanie sa skúmaného prvku, ktoré sa nejakým spôsobom dotýka vybraných vlastností. V našom prípade je to rozsvietenie alebo zhasnutie žiarovky. Týmto postupom sme v priestore riešení získali model prvku z priestoru problémov. Tento model môže slúžiť na opis klasickej žiarovky na osvetľovanie, signalizačnej žiarovky...,Nami vytvorený model opisuje celú triedu prvkov typu žiarovka.

Na základe všeobecnej tiedy môžeme vytvárať jej kokrétne "inštancie" typu žiarovka, ktoré zodpovedajú konkrétnym prvkom – objektom. Je potrebné zdôrazniť rozdiel medzi pojmami trieda a p objekt. Tieto pojmy sa často zamieňajú alebo nerozlišujú. Trieda je novovytvorený typ, zatiaľ čo objekt je premenná tohto typu. Vytvorenie nového typu si vyžaduje potrebu jeho deklarácie a definície. Definícia opisuje, ako to trieda robí. Deklarácia sa umiestňuje do hlavičkového súboru a definícia do implementačného súboru.

Diagramy tried môžeme zjednodušene chápať ako určitý druh pohľadu na kód, pričom samy kód neobsahujú. Nakoľko medzi dagramom a kódom existuje obojsmerná interakcia, stávajú sa diagramy tried súčasťou celého životného cyklu aplikácie. Do projektu aplikácie je možné pridať diagramy tried pomocoy kontextového menu projektu „Add New Item", pričom v ponuke označíme voľbu Class Diagram. Pri pohľade na diagram tried získame komplexný prehľad na kód. Filozofia tvorby diagramov a aj ich grafická vyjadrovacia schopnosť pomáhajú pri orientácii sa v zložitých objektových štruktúrach a do určitej miery kopírujú a prispôsobujú sa vlastnostiam CLR kódu.



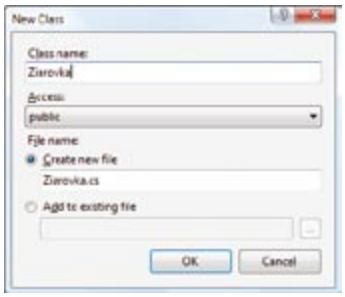
Pridanie diagramu tried do projektu

Modelovanie tried objektovo orientovanej aplikácie je do značnej miery podobný UML diagramom. Môžeme ho vytvárať pred začiatkom kódovania, napríklad v pokročilejšej fáze modelovania, alebo aj kedykoľvek

v priebehu celého životného cyklu aplikácie pre zdokumentovanie hierarchie a štruktúry tried aplikácie, alebo modulu. Diagram tried umožňuje navrhovať a využívať všetky aspekty objektovo orientovaného programovania, napríklad dedičnosť, inheritanciu.Jeden diagram môže obsahovať aj viac tried.

Z toolbaru môžeme pridávať do diagramu symboly pre triedu, abstraktnú triedu, enumerator, štruktúru, delegáta, inheritanciu, asociáciu a komentár.

Presunutím symbolu „Class" z toolboxu na plochu diagramu sa zobrazí dialóg pre pomenovanie novovytvorenej triedy. Triedu pomenujeme, napríklad v našom prípade „Žiarovka"



Dialog pre vytvorenie novej triedy

Po pridaní novej triedy do diagramu sa zobrazí prázdna šablóna pre kód. Napríklad ak pridáme do diagramu blok triedy „Potravina", bude šablóna pre kód triedy v tvare

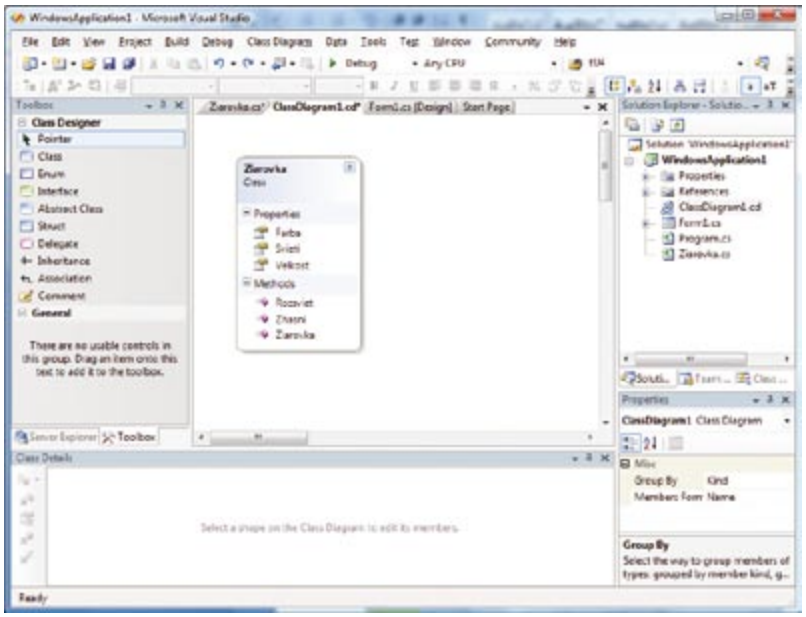
```
using System;
using System.Collections.Generic;
using System.Text;

namespace WindowsApplication1
{
    public class Ziarovka
    {
    }
}
```

Na diagrame sa zobrazí symbol triedy. V kontextovom menu symbolu môžeme pridať členskú metódu, parameter, pole, udalosť, vytvoriť konštruktor a deštruktor.

Ak pridáme pre triedu „Ziarovka" parametre „Farba", „Velkost", stavový parameter „Svieti", konštruktor triedy žiarovka a metódy Rozsviet() a Zhasni () bude vygenerovaný kód

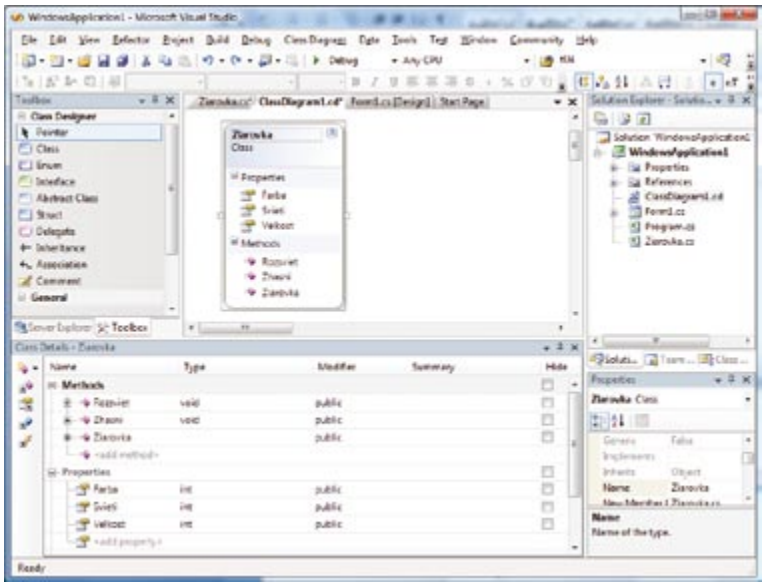
```
public class Ziarovka
    {
        public Ziarovka()
        {
            throw new System.NotImplementedException();
        }

        public int Farba
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
```

```
        }

        public int Velkost
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
        }

        public int Svieti
        {
            get
            {
                throw new System.NotImplementedException();
            }
            set
            {
            }
        }

        public void Rozsviet()
        {
            throw new System.NotImplementedException();
        }

        public void Zhasni()
        {
            throw new System.NotImplementedException();
        }
    }
```



Pracovná plocha Visual Studia 2005 v režime návrhu triedy

Metódy, parametre a procedúry pre obsluhu udalostí môžeme pridávať aj v okne „Class Details", ktoré sa nachádza v spodnej časti obrazovky vývojového prostredia.

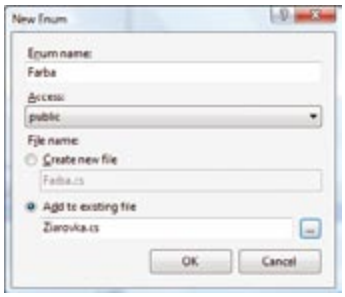


Metódy, parametre a udalosti môžeme pridávať aj v okne „Class Details"

Pre vyjadrenie hodnôt farby by sa ideálne hodil enumerátor, v ktorom by boli preddefinované všetky hodnoty parametra, teda farby, ktorou môže žiarovka svietiť.

```
enum Farba
{
    Žltá,
    Červená,
    Zelená,
    Modrá,
}
```

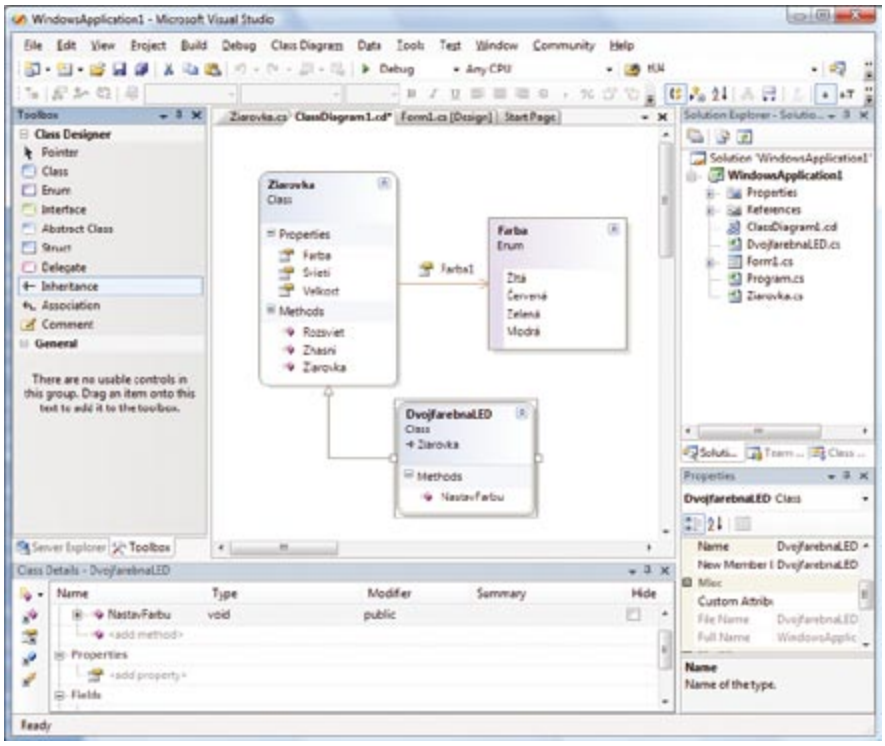
Kód pre enumerátor môžeme umiestniť do samostatného súboru, alebo pridať do súboru s kódom triedy. Keďže enumerátor sa v našom príklade používa pre vyjadrenie farby ziarovky, môžeme jeho kód pridať do súboru zo zdrojovým kódom Ziarovka.cs



Kód pre enumerátor môžeme pridať aj do súboru s kódom triedy.

Podobne ako enumerátor by sme mohli pridať do diagramu aj štruktúru, rozhranie a podobne.
Z najdôležitejších čŕt objektovo orientovaného programovania ukážeme dedičnosť. Vytvorime novú triedu „DvojfarebnaLED", ktora bude dediť všeky vlastnosti, parametre a metódy od triedy „Ziarovka". Do odvodenej triedy „DvojfarebnaLED" však pribudne nová metóda „NastavFarbu". Na tomto príklade vidíme, že táto metóda odvodenenj triedy charakterizuje novú vlastnosť objektu, ktorý je vyjadrený odvodenou triedou. Dvojfarebná led dióda má všetky vlastnosti tiredy žiarovka a navyše umožňuje zmenu farby. Pre objekty triedy žiarovka, alebo iné odvodené triedy ako napríklad Ziarivka je táto metóda irelevantná, nakoľko tieto objekty nedisponujú možnosťou zmeny farby.

Do diagramu pridáme novú triedu, pomenujeme ju „DvojfarebnaLED", vytvoríme jej metódu „NastavFarbu", pričom ako typ parametra využijeme enumerátor „Farba". Novú triedu „DvojfarebnaLED" prepojíme so základnou triedou „Ziarovka" pomocou prepojenia typu „Inheritance". Tymto grafickým prepojením definujeme, že trieda „DvojfarebnaLED" je odvodená od triedy „Ziarovka"



Trieda DvojfarebnaLED dedí vlastnosti od triedy Ziarovka

Na základe grafického návrhu vznikne kód odvodenej triedy

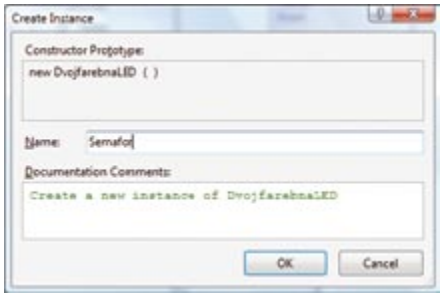
```
public class DvojfarebnaLED : Ziarovka
{
    public void NastavFarbu(Farba Farba)
    {
        throw new System.NotImplementedException();
    }
}
```

Aby sme sa presvedčili o tom, že diagram neobsahuje žiadne doplňujúce údaje a je vlastne len pohľadom na kód, môžeme urobiť jednoduchý pokus. Zmeňme názov metódy NastavFarbu(Farba Farba) na ZmenFarbu(Farba Farba). Zmena sa okamžite prejaví na diagrame. Diagram tried je teda len určitým grafickým druhom pohľadu na kód tried. Potvrdí nám to aj pohľad na obsah súboru Diagram1. CD. Tento súbor je XML dokument a obsahuje len odkazy na súbory obsahujúce kódy objektov.

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Font Name="Segoe UI" Size="9" />
  <Class Name="WindowsApplication1.Ziarovka">
    <Position X="0.5" Y="0.5" Width="1.5" />
    <TypeIdentifier>
      <FileName>Ziarovka.cs</FileName>
      <HashCode>AACAACAAAAAAAIAAAAAAAAAAAAAAAAAAAEAABAAAAAAAA=</HashCode>
    </TypeIdentifier>
    <ShowAsAssociation>
      <Property Name="Farba1" />
    </ShowAsAssociation>
  </Class>
  <Enum Name="WindowsApplication1.Farba">
    <Position X="3" Y="1" Width="1.5" />
    <TypeIdentifier>
      <FileName>Ziarovka.cs</FileName>
      <HashCode>AAAAAAAAAAAAAAAAAAAABAAAgAABgAAAAAAAAAAAAAA=</HashCode>
```

```
    </TypeIdentifier>
  </Enum>
</ClassDiagram>
```

Vývojové prostredie Visual Studio 2005 umožňuje vytvárať aj testovacie inštancie objektov v okne „Object Test Bench“. Napríklad takto môžeme vytvoriť inštanciu triedy „DvojfarebnaLED“ s názvom napríklad „Semafor“ a otestovať fungovanie tejto triedy.

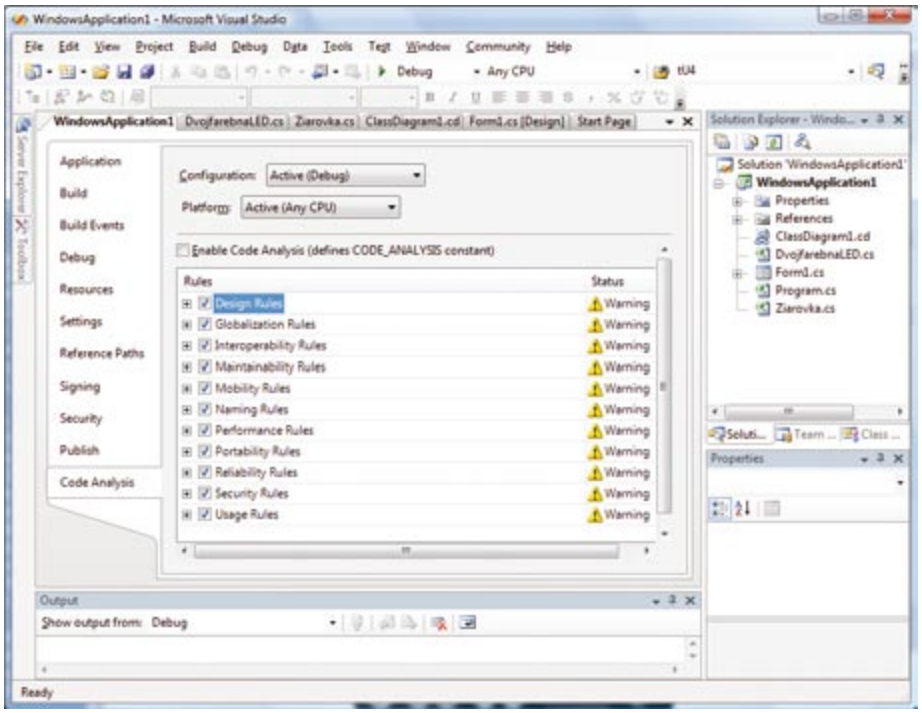


Vytvorenie testovacej inštancie triedy v Object Test Bench

KAPITOLA 4: NÁSTROJE PRE ANALÝZU KÓDU

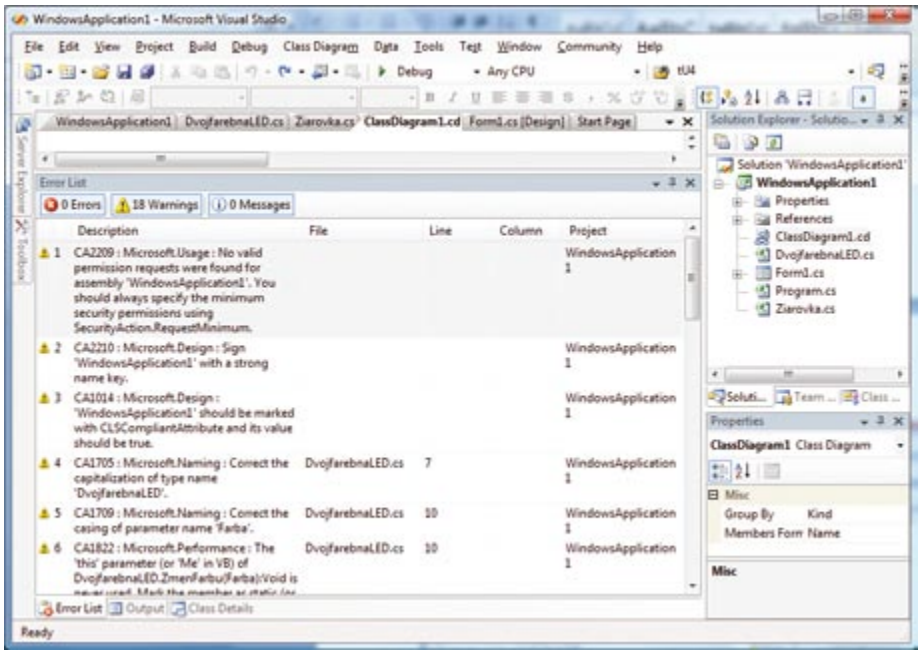
Nástroje pre analýzu umožňujú vykonávať dva druhy analýz – statickú a dynamickú. Statická analýza umožnuje analyzovať kód pred jeho spustením. Je potrebné uvedomiť si rozdiel medzi kompiláciou a statickou analýzou kódu. Úspešná vykonaná kompilácia znamená len to, že zdrojový kód neobsahuje žiadne syntaktické chyby, no ani zďaleka to neznamená, že aplikácia skutočne pobeží a že bude vykonávať požadovanú funkcionalitu. Pomocou nástrojov pre analýzu kódu integrovaných do Visual Studia 2005 (pre zaujímavosť jedná sa o nástroje predtým známe ako FxCop a PREFast) sa analyzuje úspešne skompilovaný kód. Statická analýza je integrovaná aj na úrovni Team Foundation Serveru. Pre zaistenie čo najvyššej kvality procesu tímového vývoja a v neposlednom rade aj jeho efektívnosti je možné predpísať, že na tímový server sa uloží len kód, ktorý úspešne prešiel procesom statickej analýzy, čo už samo o sebe zaisťuje určitý stupeň kvality a elimináciu niektorých potenciálnych problémov, ktoré by mohli neskôr vývoj projektu skomplikovať.

V konkrétnom projekte na ktorý chceme aplikovať statickú analýzu kódu najskôr v okne Solutions Explorer aktivujeme položku Properties pre aktuálny projekt. V okne Properties nájdeme záložku „Code Analysis" (posledná záložka v poradí) a v nej zaškrtneme, respektíve ponecháme zaškrtnuté pravidlá, ktoré chceme pri statickej analýze kódu skontrolovať.



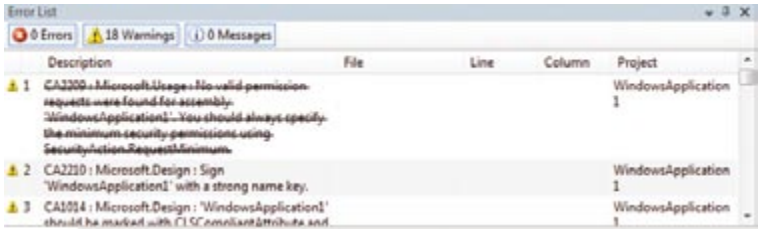
Nastavenie pravidiel pre statickú analýzu kódu

V záložke „Code Analysis" môžeme zaškrtnúť aj voľbu „Enable Code Analysis". Pri takomto nastavení sa vykoná statická analýza kódu po každej kompilácii. Po nastavení alebo upresnení pravidiel môžeme pomocou položky kontextového menu „Run Code Analysis" spustiť statickú analýzu kódu. Po jej vykonaní sa v spodnom okne zobrazí zoznam upozornení. O tom aká je statická analýza dôležitá a v neposlednom rade aj prísna svedčí aj jej výsledok v našom prípade, kde sme mali vytvorenú aplikáciu obsahujúci len kód pre triedu Ziarovka z predchádzajúcej state. Ako výsledok analýzy sme získali 18 oznamov o potenciálnych problémoch.



Zoznam upozornení po statickej analýze kódu

Ak niektoré chybové hlásenie nie sú relevantné s našou požadovanou politikou požiadaviek na aplikáciu môžeme toto hlásenie pomocou funkcie kontektového menu „Supress Message“ vypnúť, takže sa bude ingnorovať a nebuda ani prekážkou pre uloženie projektu na server pre tímovú spoluprácu.



Potlačenie niektorých hlásení po statickej analýze kódu

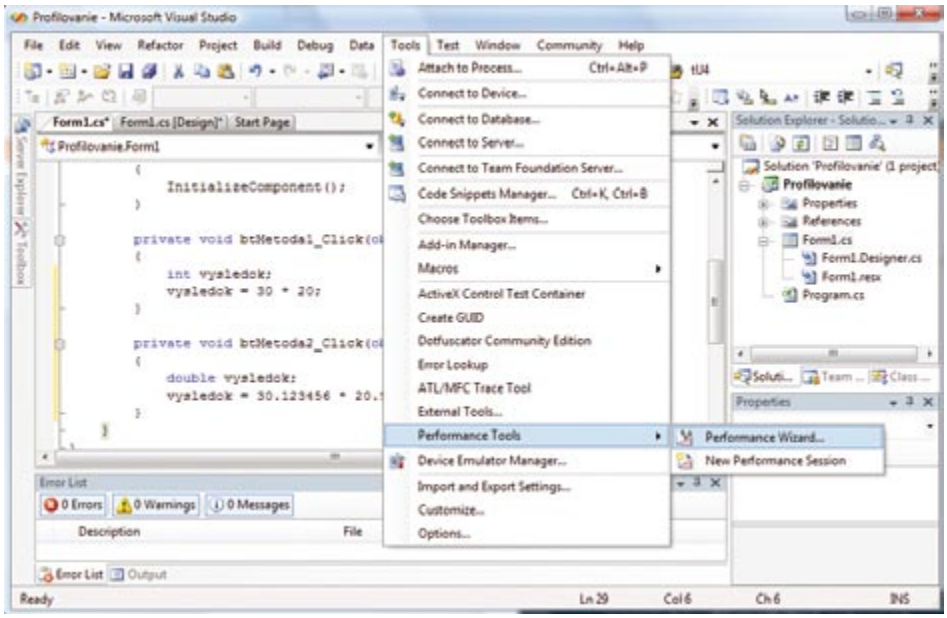
Informácie o potlačení niektorých chybových oznamov sa uloží do súboru GlobalSuppressions.cs v tvare

```
[assembly: System.Diagnostics.CodeAnalysis.SuppressMessage(„Microsoft.
Design“, „CA2210:AssembliesShouldHaveValidStrongNames“)]
```

Dynamická analýza (profilovanie) bežiaceho kódu

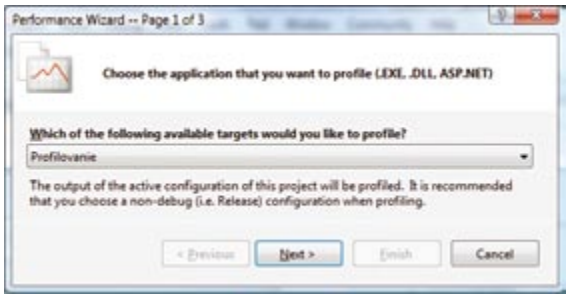
Na rozdiel od statickej analýzy, ktorej predmetom je skompilovaný kód, ktorý nie je spusteny, dynamická analýza prebieha nad spustenou aplikáciou. Môže sa aplikovať nad aplikáciami využívajúcimi buď riadený (managed) nebo nativny kód. Proces dynamickej analýzy nazývame niekedy aj profilovanie. Do analýzy je zahrnutý zber informácií o alaokácii pamäti, procesu dynamického vytvára nia objektov a ich zániku, analýza využívania zásobníka, rozbor času, ktoré potrebujú pre svoj beh jednotlivé bloky kódu a podobne. Dynamická analýza môže vykonávať vzorkovanie údajov v pravidelných intervaloch, alebo pri volaní jednotlivých blokov kódu. Vzorkovanie v pravidelných intervaloch je hardvérovo závislé, deje sa dokonca až na úrovni procesora preto sa nedá vykonávať na virtuálnych počítačoch.

Nástroj „Performance Wizzard“ pre profilovanie aplikácie aktivujeme z menu Tool, Performance Tools. Nastavenie profilovania sa vykoná v troch krokoch



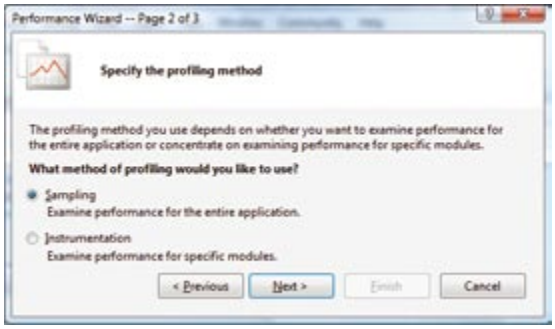
Zapnutie dynamickej analýzy (profilovania)

V prvom kroku sprievodcu „Performance Wizzard" vyberieme aplikáciu, ktorú chceme analyzovať a profilovať



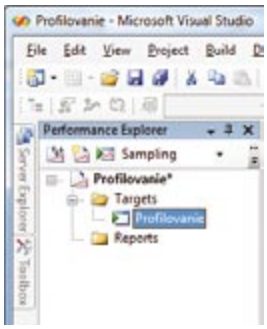
„Performance Wizzard", krok 1 – výber aplikácie

V druhom kroku vyberieme metódu profilovania. V našom prípade sme zvolili hardvérové vzorkovanie v pravidelných intervaloch na úrovni procesora

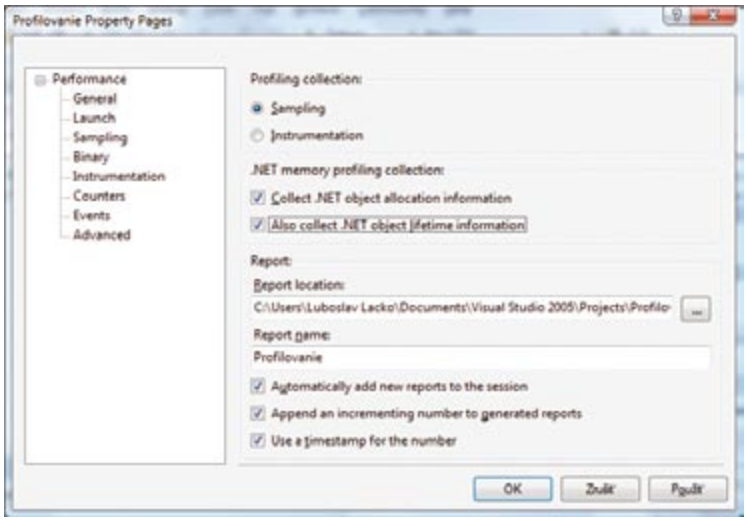


„Performance Wizzard", krok 2 – výber metódy vzorkovania

Tretí krok sprievodcu „Performance Wizzard" je potvrdzovací. Po potvrdení nastavenia profilovania sa v ľavej časti pracovnej plochy Visual Studia zobrazí nástroj „Performance Explorer". Profilovanie zahájime tlačidlom „Launch" (tlačidlo na ktorom je ikona grafu so zelenou šípkou)

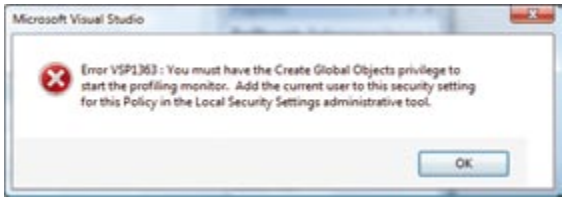


Performance Explorer

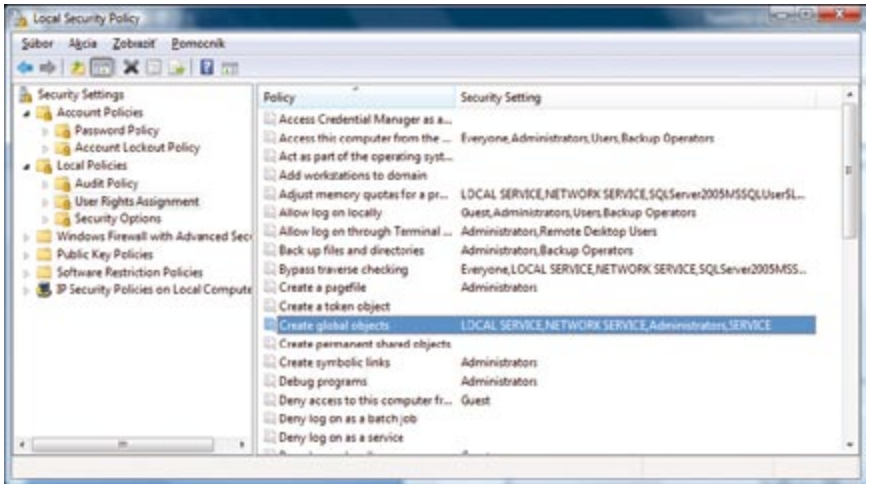


Dialóg pre nastavenie vlastnosti profilovania

Aby bolo možné spustiť profilovanie aplikácie, bude na klientskom vývojárskom počítači v niektorých prípadoch potrebné nastaviť pre aktuálneho používateľa privilégium „Create Global Objects“



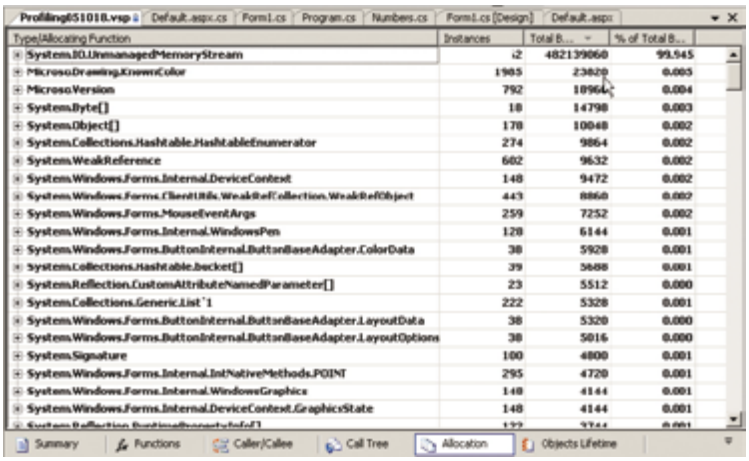
Pre profilovnaie musíme nastaviť príslušné privilégiá na úrovni operačného systému



Nastavenie privilégia „Create Global Objects“

Po spustení aplikácie vykonáme príslušné používateľské úkony tak aby sme aktivovali tie bloky kódu ktoré chceme profilovať. O výsledku profilovania získame report, ktorý podľa toho čo chceme profilovať bude obsahovať názvy typov a funkcií, ktoré najviac obsadzujú pamäť, zoznamy objektov a typov, pre ktoré sa vytvorí najviac inštancií, doba trvania bloku kódu a podobne. Report o výsledkoch profilovania je prehľadne rozdelený do záložiek

- Summary
- Functions
- Celler/Calle (znázornosti postupnosti volania funkcií)
- Call Tree
- Allocation
- Objects Lifetime



Reporty o výsledku profilovania

KAPITOLA 5: NÁSTROJE PRE TESTOVANIE KÓDU

Ku kvalite vyvíjaného kódu prispieva nielen statická a dynamická analýza kódu, ale v zmysle hesla „Dôveruj ale preveruj" najmä jeho dôkladné otestovanie. Nástroje pre testovanie kódu boli do verzie Visual Studio 2005 veľmi tesne integrované. Pri testovaní nie je najvýznamnejší jeho technický aspekt, teda prepracovanosť testovacích procedúr, ale najmä filozofia voľby pokrytia kódu a rozsahu parametrov. Koncepcia testovania vychádza z predpokladu, že sa vytvárajú testovacie metódy založené na atribútoch, ktoré vedú k úspechu testu. Testovacie triedy sú generované automaticky. Kód testovacieho projektu sa spravuje a verziuje úplne rovnako ako akýkoľvek iný zdrojový text aplikácie. V prípade zlyhania testu sa automaticky generuje chybová správa (bug). Keď tím vývojárov vytvorí na tento bug záplatu, táto záplata, alebo komplexnejší opravný balíček sa automaticky zahrnie do novej verzie testovacieho procesu.

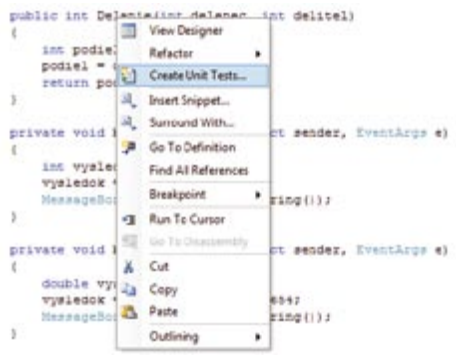
Pre čo najvyššiu účinnosť testovania sa odporúča umiestniť testy do samostatných projektov, čiže ku každému projektu, ktorý chceme testovať vytvoríme jeho testovací projekt. Dôležité je nielen testovací projekt vytvoriť, ale ho aj udržiavať. To znamená, že ak sa v životnom cykle vývoja projektu do kódu pridá nová metóda, je potrebné do testovacieho projektu zahrnúť aj kód pre otestovanie tejto novej metódy. Podobne ako nie je dobré aby programátori písali ku kódu ktorý vtvorili návody, nemali by vytvárať ani testy. Ideálna koncepcia pre tvorbu testov je koncepcia „čiernej skrinky", kedy nás nezaujíma implementácia, ale požiadavky na funkcionalitu. Testy vytvárame na základe zadania a jeho analýzy a to najlepšie ešte v etape analýzy zadania a návrhu riešenia. Takéto testy sú oveľa účinnejšie ako testy, ktoré napíšeme po ukončení vývoja, kedy sa podvedome snažíme prispôsobiť testy implementácii zadania. Testy by mali byť prehľadné, to znamená, že každý test by mal testovať len jednu funkciu alebo metódu a mal by byť nezávislý na ostatných testoch. Aj napriek odporúčaniu nevytvárať testy podľa konkrétnej implementácie spravidla dokážeme definovať hraničné podmienky a oblasť zakázaných hodnôt, napríklad podľa matematickej teórie, alebo princípov fungovania objektívnej reality, podľa ktorej vytvárame model aplikácie. Použijeme trochu úsmevný ilustračný príklad – ak vytvárame textovú adventúru „Snehulienka a sedem trpaslíkov", z objektívnej reality by malo vyplývať, že ôsmy a každý ďalší trpaslík je logickou chybou. Testujme teda aj hraničné prípady a prípady, ktoré vedú k očakávaným výnimkám.

Aplikáciu spravidla testujeme rôznymi nezávislými metódami a tímom nezávislých beta testerov. Ak niektorý beta tester objaví nejakú chybu, najskôr vytvoríme pre túto chybu test, ktorý ju spoľahlivo identifikuje a až potom ju začneme opravovať.

Ako príklad pre demonnštráciu testovania vytvoríme jednoduchú funkciu, ktorej úlohou bude vydeliť dva čísla.

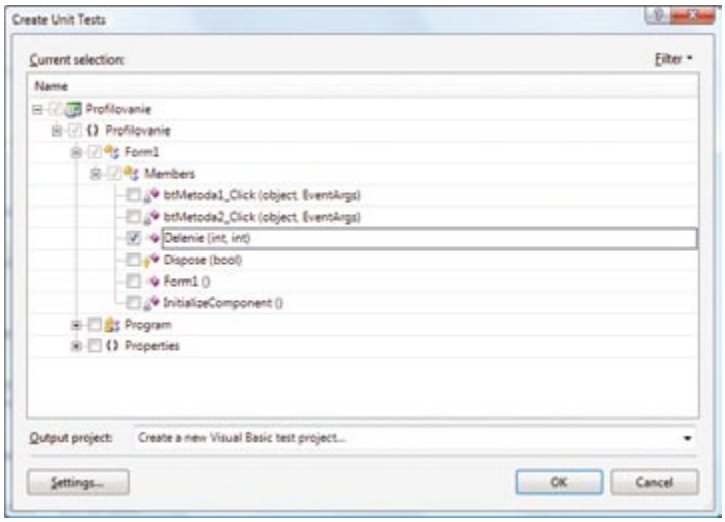
```
public int Delenie(int delenec, int delitel)
{
    int podiel;
    podiel = delenec/delitel;
    return podiel;
}
```

Testovací unit pre danú metódu, funkciu alebo procedúru vytvoríme pomocou položky kontextového menu „Create Unit Tests"



Aktivovanie vytvorenia testovacieho unitu pre procedúru

V dialógu „Create Unit Tests“ vyberieme pre ktoré jednotky kódu je potrebné vygenerovať testovacie metódy. V našom príklade sa bude jednať o metódu Delenie()



Dialóg pre vytvorenie testovacieho unitu

Na záver testovací projekt nejako pomenujeme. V testovacom projekte bude pre metódu delenia vytvorená testovacia metóda `Public Sub DelenieTest().`

```
  ,''<summary>
    ,''A test for Delenie (int, int)
    ,''</summary>
    <TestMethod()> _
    Public Sub  ,''<summary>
    ,''A test for Delenie (int, int)
    ,''</summary>
    <TestMethod()> _
    Public Sub DelenieTest()
        Dim target As Form1 = New Form1

        Dim delenec As Integer ,TODO: Initialize to an appropriate value

        Dim delitel As Integer ,TODO: Initialize to an appropriate value

        Dim expected As Integer
        Dim actual As Integer

        actual = target.Delenie(delenec, delitel)

        Assert.AreEqual(expected, actual, „Profilovanie.Form1.Delenie did
not return the expected value.“)
        Assert.Inconclusive(„Verify the correctness of this test method.“)
    End Sub
```

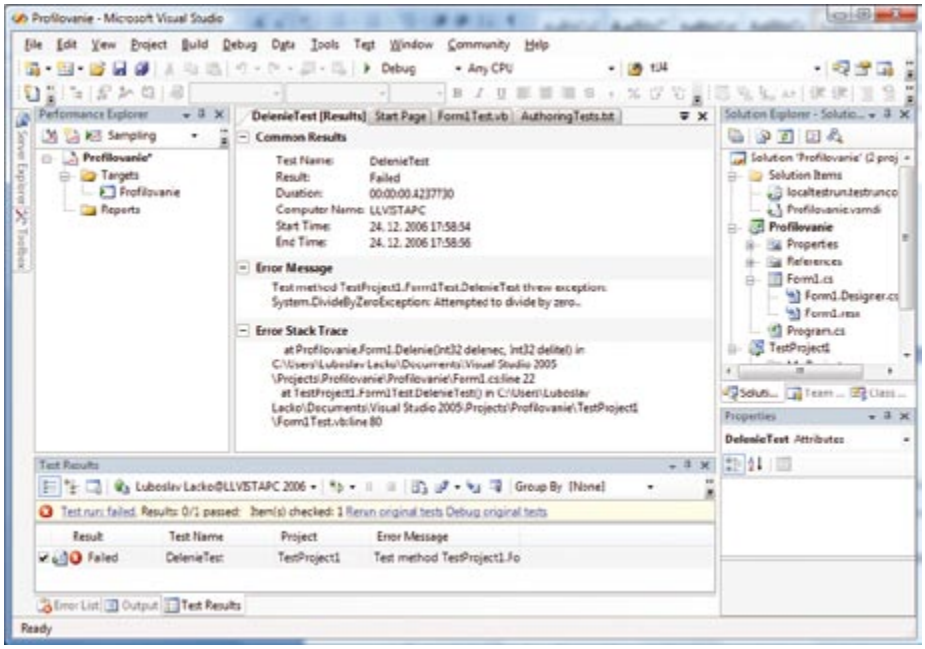
Samozrejme v prípade delenia nás ihneď napadne „zlomyselné“ nastavenie počiatočných parametrov. Najskôr však vyskúšajme prípustné hodnoty

```
        Dim delenec As Integer = 20
        Dim delitel As Integer = 2
```

Môžeme nastaviť aj hodnotu očakávaného výsledku (expected) na hodnotu 10. Test prebehol úspešne. Teraz skúsme deliť nulou, preto hodnoty parametrov „delenec“ a „delitel“ nastavme takto

```
        Dim delenec As Integer = 20
        Dim delitel As Integer = 0
```

Pri testovaní samozrejme táto chyba bude okamžite odhalená a identifikovaná



Identifikácia chyby odhalenej pri testovaní

Ak chceme testovat funkciu na rozsiahlejšiu množinu vstupných parametrov, môžeme namiesto priameho zadania hodnôt zadať pripojovacie parametre na databázovú tabuľku, ktorá množinu testovacích hodnôt obsahuje

```
        [DataSource(„System.Data.SqlClient“, „Data Source=TESTY;Integrated
Security=True“, „tblDelenie“, DataAccessMethod.Sequential)]
        [TestMethod()]
        public void DelenieTestDBRun()
        {
            int delenec = (int)TestContext.DataRow[„delenec“];
            int delitel = (int)TestContext.DataRow[„delitel“];
            int expected = (long)TestContext.DataRow[„podiel“];
            int actual;
            actual = target.Delenie(delenec, delitel)
...
...
```

Pokyny pre prácu s atribútmi `ClassInitialize`, `ClassCleanup`, `TestInitialize` a `TestCleanup` pri testovaní nájdeme v komentároch v regióne kódu `„Additional test attributes“`

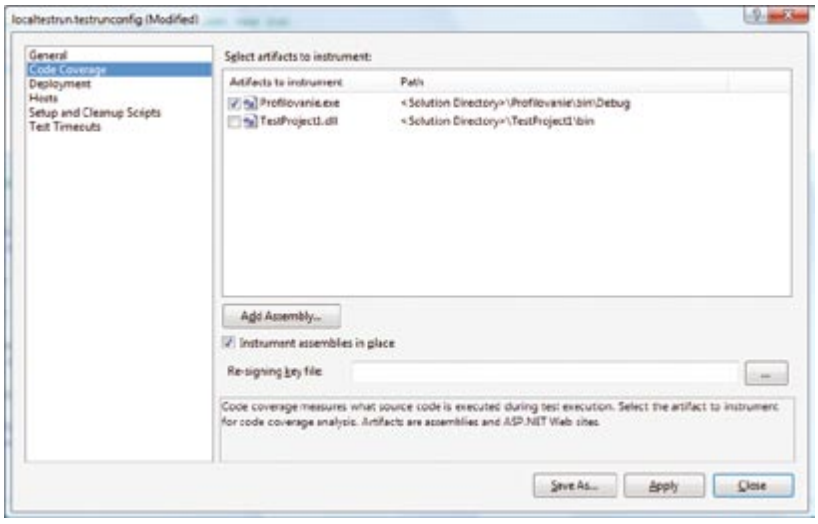
```
,You can use the following additional attributes as you write your tests:
,
,Use ClassInitialize to run code before running the first test in the class
,
,<ClassInitialize()>  _
,Public Shared Sub MyClassInitialize(ByVal testContext As TestContext)
,End Sub
,
,Use ClassCleanup to run code after all tests in a class have run
,
,<ClassCleanup()>  _
,Public Shared Sub MyClassCleanup()
```

```
,End Sub
,
,Use TestInitialize to run code before running each test
,
,<TestInitialize()>  _
,Public Sub MyTestInitialize()
,End Sub
,
,Use TestCleanup to run code after each test has run
,
,<TestCleanup()>  _
,Public Sub MyTestCleanup()
,End Sub
,
```

Aj napriek tomu, že pre testovanie je oveľa dôležitým kritériom kvalita testovania, je potrebné pokryť čo najväčší rozsah kódu, pričom si ale netreba klásť nerealistické ciele, pretože stopercentné pokrytie zložitejšej aplikácie kvalitnými testami je prakticky nedosiahnuteľné. Pokrytie kódu testovacími unitami je možné merať pomocou kritéria „code coverage". Pri plánovaní a realizácii pokrytia kódu testovacími unitami postupujeme zhora nadol v zmysle hierarchického poradia

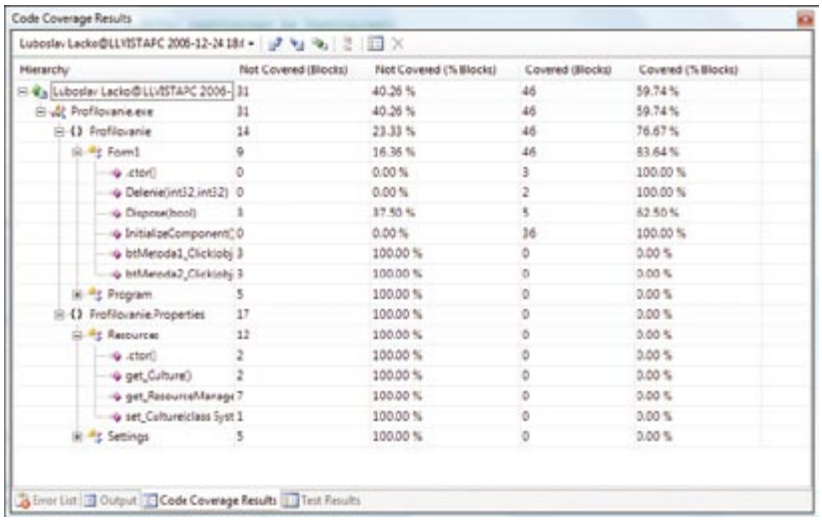
systém -> riešenie -> projekt -> trieda -> metóda...

Ak chceme merať pokrytie kódu testovacími unitami pomocou meny Test – Edit Test Run Configurations povolíme v záložke „Code Coverage" meranie pokrytia kódu v položkách projektu



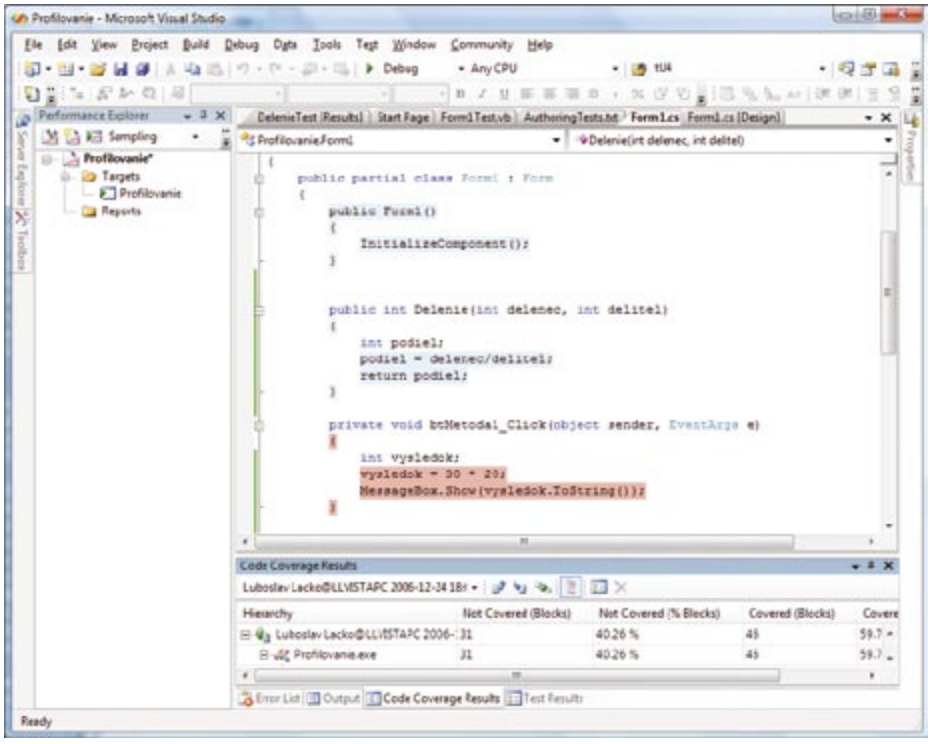
Zapnutie merania pokrytia kódu pri testovaní

Vidíme, že je pokrytých takmer 60 percent kódu.



Zobrazenie výsledkov merania pokrytia kódu testovacími unitami

Výsledky merania pokrytia kódu testovacími unitami môžeme zobraziť aj graficky priamo v zdrojovke ak v okne „Code Coverage Results" klikneme na názov testovanej metódy. Pokrytý kód sa zobrazí so zeleným podfarbením a nepokrytý kód bude podfarbený na červeno

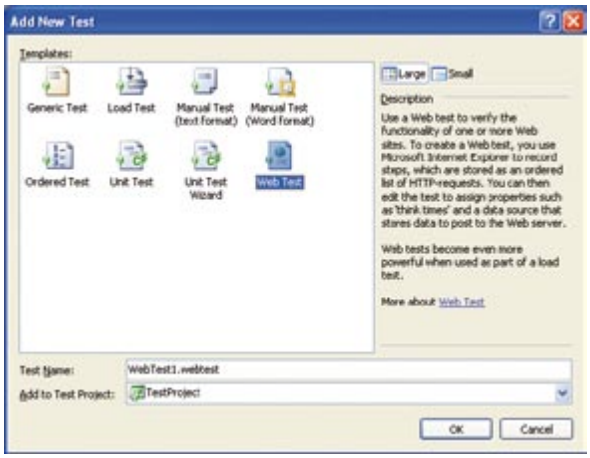


Výsledky merania pokrytia kódu testovacími unitami môžeme zobraziť aj graficky priamo v zdrojovke

KAPITOLA 6: FUNKČNÉ TESTY WEBOVÝCH APLIKÁCIÍ

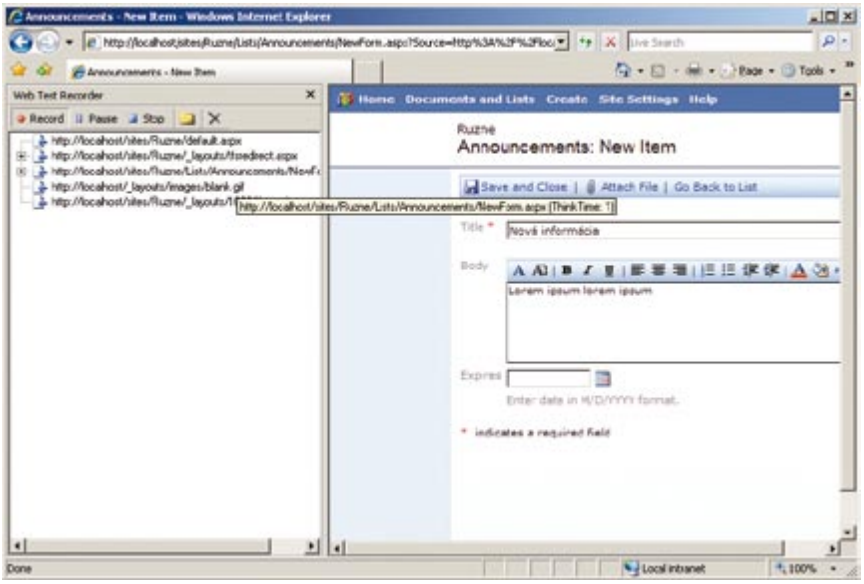
Úloha otestovať webovú aplikáciu je veľmi náročná, pretože u webových aplikácií, na rozdiel od aplikácií typu WinForms vstupujú do hry ďalšie faktory, napríklad typ a verzia webového prehliadača u klienta, nastavenie niektorých parametrov, napríklad cookies a podobne. Kapitolou samou o sebe je dnešný trend číslo jedna – ktorým je bezpečnosť a z bezpečnostných požiadaviek vyplývajúce zabezpečenie počítača. Na jednej strane, čím viac toho pre webové aplikácie klient zakáže, tím bude mať väčší pocit bezpečia, no na druhej strane mu nemusia fungovať niektoré aktívne prvky webových aplikácií. Z naznačeného okruhu problémov je zrejme, že automatizované testovanie v tomto prípade ani len zďaleka nenahradí testovanie testermi – teda živými osobami. Niektoré okruhy problémov však môžeme s výhodou otestovať pomocou automatizovaných tesov, napríklad aká je doba odozvy aplikácie na typickú úlohu, alebo či dáva vyhľadávanie v databázových aplikáciách očakávané výsledky. Ak totiž ovládacie prvky pre vyhľadávacie kritériá umožňujú zostaviť síce syntaktycky aj logicky správny dotaz do databázy pre vyhľadávanie, no ak výsledkom je nájdenie státisícov záznamov, ktoré nemá zmysel vypisovať do kontextu webovej stránky, takáto aplikácia môže mať dobu odozvy aj niekoľko hodín, zaťaží sieť a v konečnom dôsledku aj tak výsledok nič neprinesie, pretože získame státisíce záznamov, ktoré nemôžeme nijako využiť (jedine s výhodou pri zlomyselnom útoku tohoto typu ukradnúť) ale nezískame požadovanú informáciu.

Vo verzii Visual Studio 2005 for Software Testers môžeme vykonávať funkčné testy webových aplikácií pomocou integrovaného nástroja, ktorý bol predtým dodávaný pod názvom „Application Center Test". Nástroj umožňuje funkčné a záťažové testovanie.



Druhy testov dostupné vo verzii Visual Studio 2005 for Software Testers

Nový test webovej aplikácie postupne definujeme tak, že po vytvorení nového projektu tohoto typu vo Visual Studiu 2005 sa v pracovnom okne vývojového prostredia otvorí prehliadač webového obsahu Internet Explorer. V ňom prechádzame v postupne po jednotlivých stránkach webovej aplikácie. Naša činnosť sa v tomto režime zanamenáva, takže postupným cieľavedomým „surfovaním" vlastne zadávame postupnosti URL adries stránok, ktoré budú predmetom testovania. Zaznamenávané URL adresy sa zobrazujú v okne „Web Test Recorder", ktoré je umiestnené v ľavej časti pracovnej obrazovky vývojového prostredia. Počasa tohoto procesu sa zachytia aj parametre, ktoré sú v rámci URL adries a HTTP protokolu prenášané. Z toho vyplýva, že web testy na platforme Visual Studia môžeme nasadiť pre akékoľvek webové aplikácie a webové služby, nielen pre technológiu ASP.NET



Nahratie web testu

Po zaznamenaní postupnosti URL adries umožňuje vývojové prostredie meniť validačné a extrakčné pravidlá, pridávať vlastné pravidlá, upravovať parametre testov, vrátane parametrov Data – driven testov, uložených v databázových tabuľkách. Sprievodcovia integrovaní vo Visual Studiu pomáhajú pri generovaní kódov pre príslušné testy. Extrahovať je možné údaje z hlavičiek stránok, textov, údaje z polí webových formulárov vrátane polí skrytých. Validáciu výsledkov je možné vztiahnuť na hodnotu polí, kontrolu počtov a spárovania tagov a kontrolu hodnôt atribútov.

Web test je v podstate postupnosťou URL adries a parametrov vyextrahovaných z týhto adries, cookies, HTTP komunikácie, query stringov a podobne. Fyzicky sú tieto údaje uložené vo forme XML dokumentu. Aj napriek tomu, že tento formát je pomerne prehľadný, parametre testov sa needitujú v tomto textovom XML dokumente, ale komfortne a bezpečne vo vizuálnom návrhovom prostredí.

Príklad fragmentu web testu

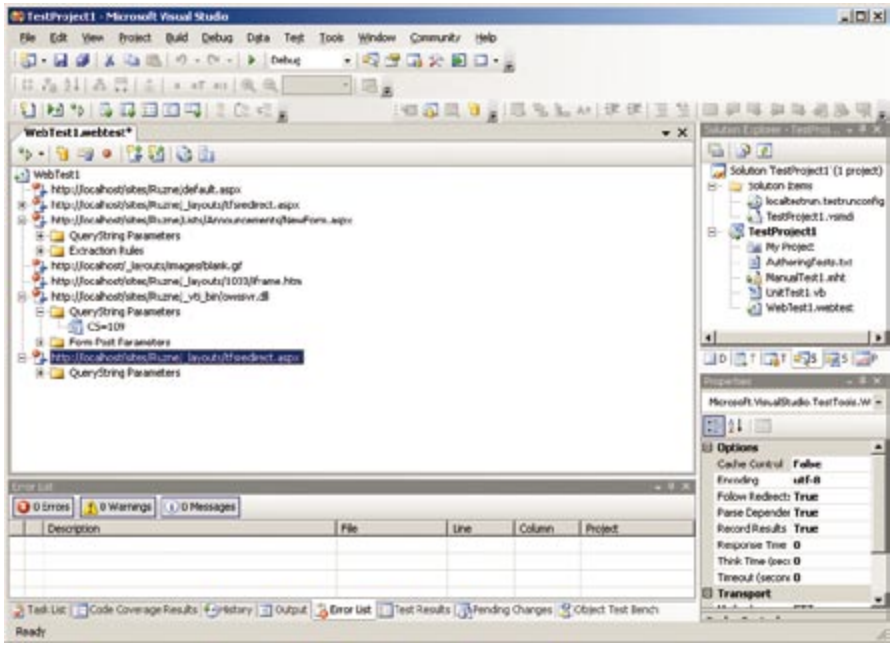
```
...
...
<Request Method="GET" Version="1.1" Url="http://localhost:2847/
KatalogZbozi/Default.aspx" ThinkTime="10" Timeout="0"
ParseDependentRequests="True" FollowRedirects="True" RecordResult="True"
Cache="False" ResponseTimeGoal="0" Encoding="utf-8" />
    <TransactionTimer Name="VlastniNakup">
      <Items>
        <Comment CommentText="Položka je vybraná náhodně" />
        <Request Method="GET" Version="1.1" Url="http://
localhost:2847/KatalogZbozi/objednat.aspx" ThinkTime="5" Timeout="0"
ParseDependentRequests="True" FollowRedirects="True" RecordResult="True"
Cache="False" ResponseTimeGoal="0" Encoding="utf-8">
          <ValidationRules>
            <ValidationRule Classname="Microsoft.VisualStudio.TestTools.
WebTesting.Rules.ValidationRuleFindText, Microsoft.VisualStudio.
QualityTools.WebTestFramework, Version=8.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" Level="High">
              <RuleParameters>
                <RuleParameter Name="FindText" Value="{{TestovaciDatabaze.
tblProduktyTest.Nazev}}" />
                <RuleParameter Name="IgnoreCase" Value="False" />
                <RuleParameter Name="UseRegularExpression" Value="False" />
                <RuleParameter Name="PassIfTextFound" Value="True" />
              </RuleParameters>
```

```
            </ValidationRule>
          </ValidationRules>
          <ExtractionRules>
            <ExtractionRule Classname="Microsoft.VisualStudio.
TestTools.WebTesting.Rules.ExtractHiddenFields, Microsoft.VisualStudio.
QualityTools.WebTestFramework, Version=8.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" VariableName="1" />
...
...
```

Podobne ako u testov z predchádzajúcej kapitoly, môžeme aj u web testov nielen meniť hodnoty parametrov tak, aby sme pokryli celú množinu očakávaných a rizikových hodnôt, ale taktiež čerpať hodnoty parametrov a očakávané výsledky z databázových tabuliek a vytvárať tak „data – driven" testy. Stačí len zachytený parameter napojiť na príslušnú databázovú tabuľku. K údajom môžeme pristupovať buď sekvenčne v takom poradí ako sú údaje uložené v databázovej tabuľke, alebo náhodne

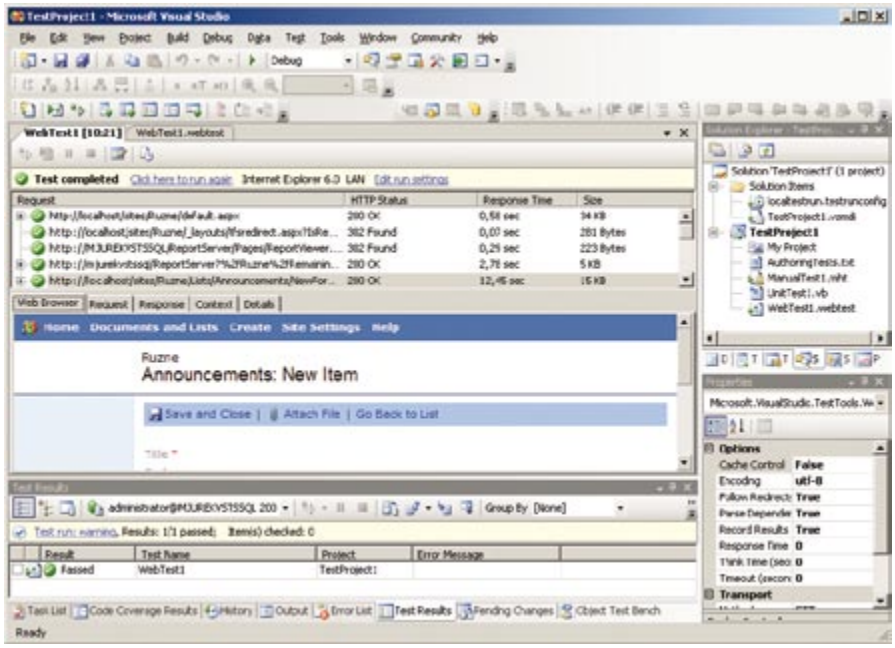
XML dokument so zaznamenaným web testom môžeme po editovaní parametrov použiť buď priamo na testovanie, alebo ho môžeme previesť na kód v .NET jazyku (C# alebo Visual Basic). Následne môžeme do takto vzniknutej šablóny kódu obsahujúcej údaje z web testu doprogramovať prakticky akúkoľvek testovaciu alebo validačnú funkcionalitu

Po ukončení záznamu surfovania testovanou aplikáciou sa vo vývojovom prostredí zobrazí zoznam URL adries. Všimnite si, že má hierarchickú stromovú štruktúru. Pri niektorých adresách sú ikonky so znakom (+), ktorý nám k príslužnej URL adrese zobrazí parametre, extrakčné pravidlá a podobne.



Zobrazenie web testu v prostredí Visual Studio 2005. Všimnite si hierarchickej štruktúry pre niektoré RL adresy

Po spustení testu dôjde v princípe k monitorovanej rekonštrukcii našich aktivít, ktoré sme vykonávali v danej aplikácii pri zadávaní testu. Z výsledkov testov pre jednotlivé URL adresy a množiny parametrov nás zaujíma nielen úspešnosť ich priebehu, ale v príslušných oknách Visual Studia môžeme prehliadať jednotlivé HTTP požiadavky, odpovede, dobu odozvy jednotlivých požiadaviek vrátane vnorených prvkov ako sú napríklad opbrázky a iné multimediálne súbory a podobne.

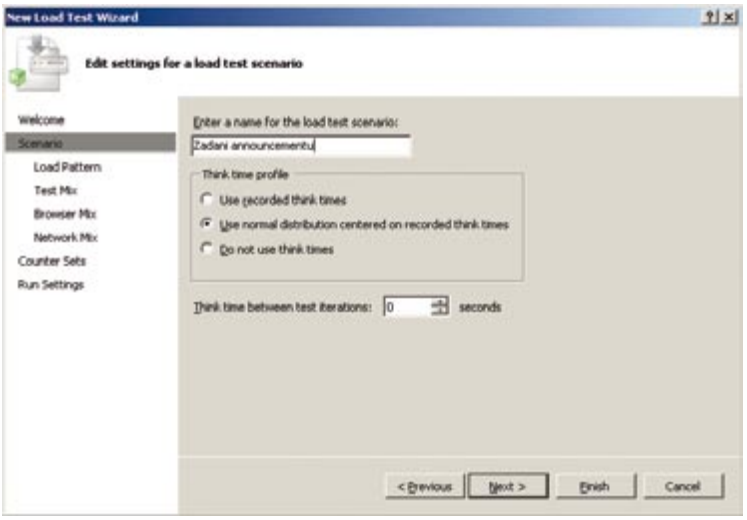


Spustenie web testu v prostredí Visual Studio 2005

KAPITOLA 7: ZÁŤAŽOVÉ TESTY

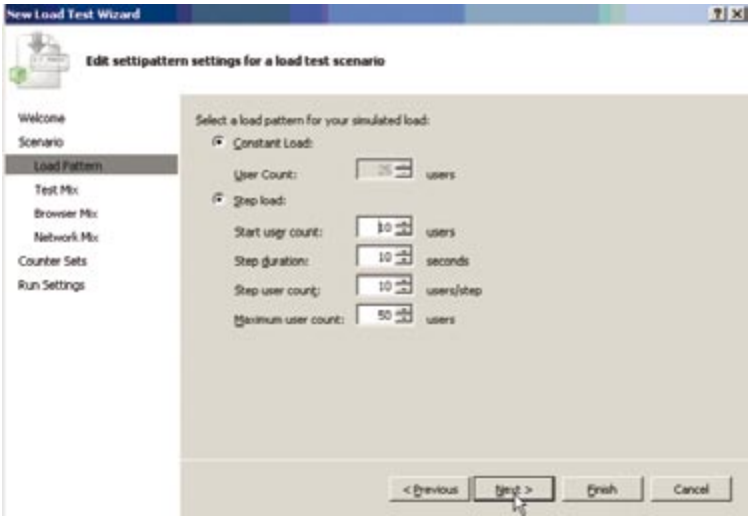
Záťažové testovanie spočíva v spúšťaní unit testov alebo web testov, ktoré vykonávame buď na viacerých počítačoch, alebo ako viaceré thready na jednom počítači. Skôr než začneme záťažové testovanie vykonávať, je potrebné nastaviť medzné hodnoty čítačov na hodnoty, ktoré považujeme pre beh príslušnej aplikácie za ešte prijateľné.

Záťažové testy môžu poztávať z jedného alebo viacerých testovacích scenárov. Pri vytváraní nového záťažového testu vytvárame pomocou sprievodcu „New Load Test Wizzard" postupne jeho jednotlivé scenáre. Prvý krok sprievodcu „Welcome" je informatívny. V druhom kroku „Scenario" najskôr vytváraný testovací scenár pomenujeme a nastavíme časoví interval medzi jednotlivými iteráciami testu



Nastavenie pre „Load test"

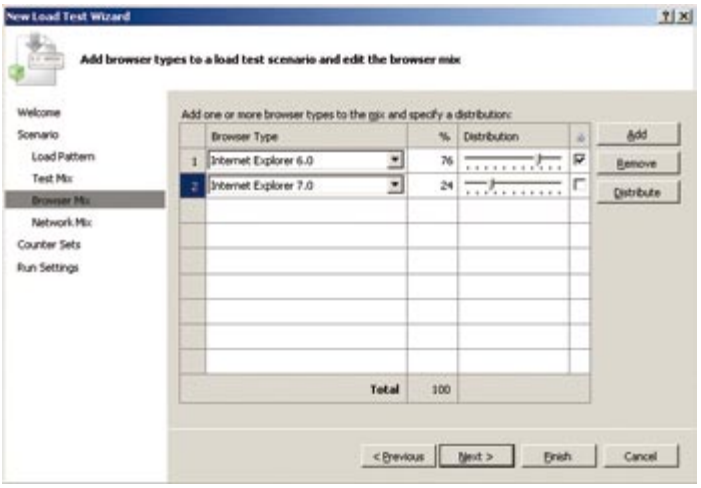
V ďalšom kroku nastavíme záťaž, ktorá je u webovej aplikácie daná počtom používateľov. Môžeme nastaviť buď konštantnú záťaž, napríklad 25 používateľov, alebo počet používateľov môže rásť od nejakej počiatočnej po koncovú hodnotu. Nastavujeme aj časový priebeh nárastu záťaže po jednotlivých krokoch, takže nastavujeme dĺžku trvania kroku a nárast počtu používateľov. V našom príklade začíname od 10 používateľov až po konečnú hodnotu záťaže 50 používateľov, pričom každých 10 sekúnd testovania sa záťaž zvýši o 10 používateľov.



Nastavenie počtu používateľov a postupnosti nárastu tohoto počtu

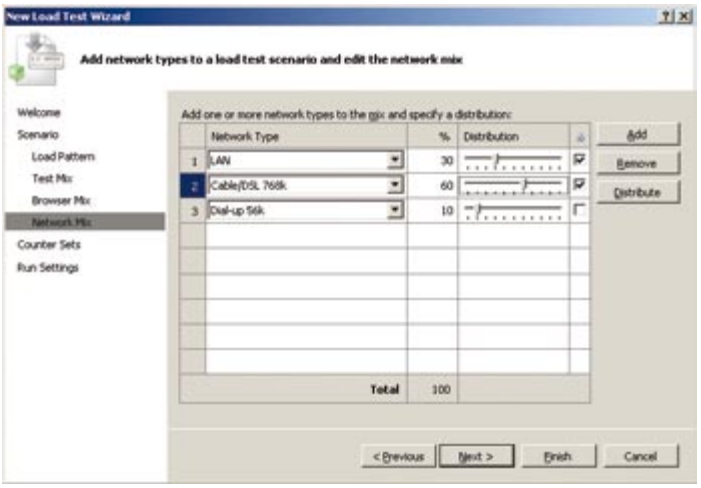
Pre záťažové testy je možné nastaviť „mix" testovacích parametrov, „mix" prehliadačov prostredníctvom ktorých sa klientsky pristupuje k testovaným webovým aplikáciám, „mix" parametrov a kvality sieťového spojenia pre aplikácie typu klient – server testované v sieťovom prostredí

Napríklad ak testujeme aplikácie tvoriace informačný systém obchodného domu, tento pozostáva z viacerých podsystémov, počnúc fotobuňkou počítajúcou vstupujúcich zákazníkov, cez skladové systémy až po systémy pokladní. Pre každý podstystém napíšeme samostatné testy a následne musíme nastaviť ich pomer tak aby odrážal reálne správanie sa zákazníka. Napríklad v hypermarkete viac než 80 percent zákazníkov si niečo kúpi alebo inými slovami – málokto odchádza z hypermarketu bez nákupu. Naproti tomu v kníhkupectve si kúpi knihu len približne 20 percent zákazníkov, takže 80 percent zákazníkov si knihy popozerá, polistuje a odídu bez nákupu. Podľa takéhoto predpokladu správana zastavujeme aj pomer záťaže, takže pre testy informačného systému hypermarketu na 100 percent testovacích priebehov testu počítania zákazníkov bude 80 percent priebehu testu pokladničného systému. U informačného systému kníhkupectva nastavíme pre test pokladničného systému len 20 percentný pomer záťaže. Podobne môžeme postupovať aj u webového testu aplikácie internetového obchodu. Na 100 percent záťaže testu prehliadania si katalógu nastavíme 15 percentný pomer záťažového testu pridania položky do košíka a elektronickej platby. Podobne nastavíme mix pomeru typov a verzií prehliadačov, napríklad 80 percent Internet Explorer, 15 percent Netscape a 5 percent záťaže bude tvoriť prístup z prehliadačov mobilných zariadení.



Výber typu prehliadača

Podobne nastavujeme mix pomerov druhu, rýchlosti a kvality sieťového pripojenia pre testovanie danej aplikácie (LAN, dial-up, GSM...). Toto nastavenie slúži ako podklad pre simulovanie oneskorení, ktoré vyplývajú z prenosu údajov prostredníctvom spojení s nižšou prenosovou rýchlosťou.

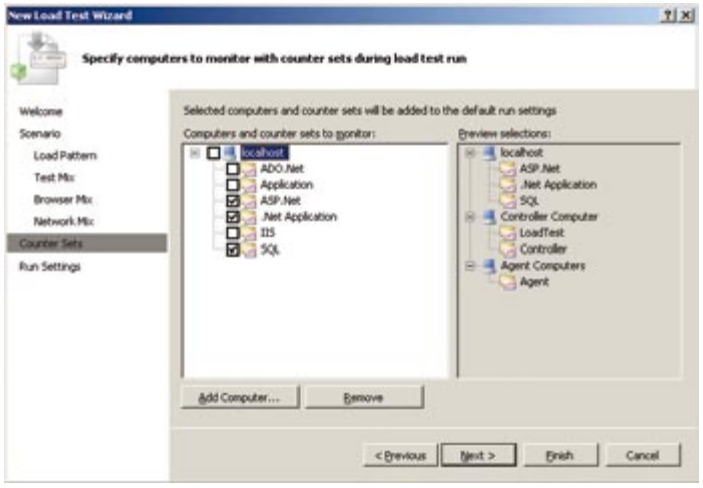


Nastavenie typu a parametrov pre sieťové prostredie

Samotnú záťaž, ako hlavný parameter záťažových testov môžeme nastaviť buď konštannú, aleboi môžeme záťaž postupne zvyšovať až po určitú predpísanú hodnotu, alebo môžeme záťaž zvyšovať dovtedy, kým nebude splnené nejaké kritérium. Taktiež je možné nastaviť pre začiatok určitú obmedzenú „zahrievaciu" záťaž a zvýšiť ju až po nejakej dobe. Pýtate sa načo je to dobré, veď softvér

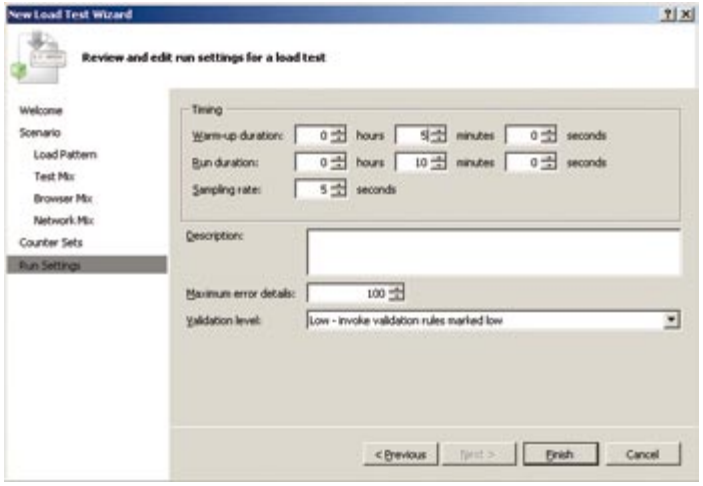
nie je spaľovací motor aby ho bolo nutné pre dosiahnutie optimálneho výkonu najskôr zahrievať? Zahrievacie kolo môže byť dobré napríklad pre ASP.NET aplikácie, kedy sa vlastne kompilácia kódu každej stránky vykonáva až pri prvom prístupe na príslušnú stránku. Takto sa potom v procese „zahrievania" prekompilujú všetky stránky tvoriace aplikáciu a samotné testy potom už prebiehajú s kompilovaným kódom, takže proces prípadnek počiatočnej kompilácie nemá vplyv na záťažové testovanie.

V kroku „Counter Sets" nastavíme ktoré čítae chceme monitorovať, v našom prípade sme zadali, že na lokálnom počítači chceme monitorovať nielen čítače pre .NET aplikácie všeobecne, ale samostatne aj čítače špeciálne pre ASP.NET aplikácie a čítače pre SQL prístup k databázovému serveru

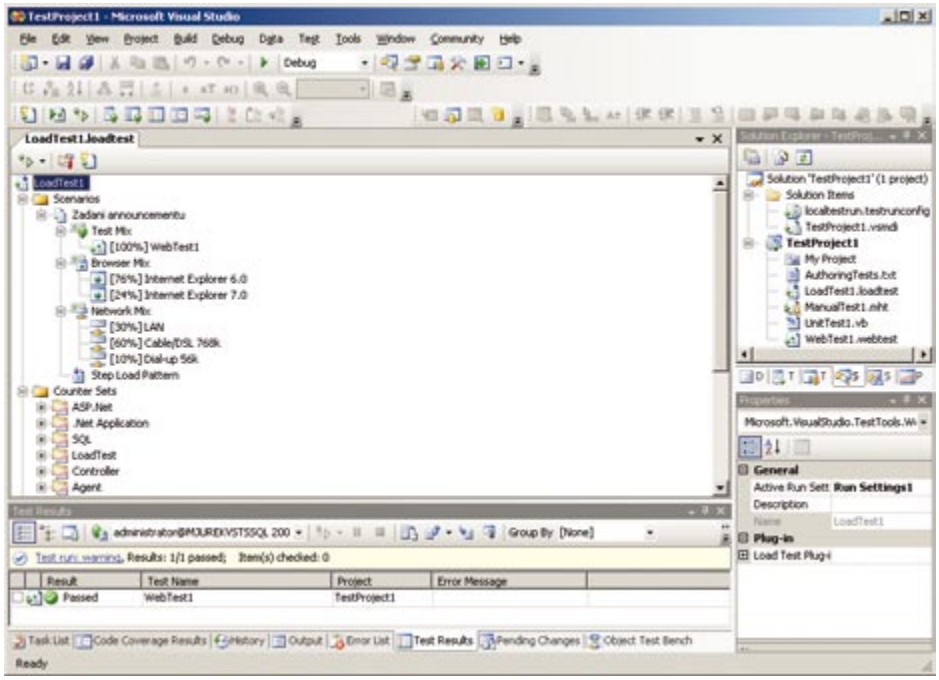


Nastavenie čítačov pre monitorovanie

V záverečnom kroku „Run Setting" nastavíme čas zahrievania, čas ako dlho chceme aby aplikácia bežala a periódu intervalu vzorkovania. V našom prípade sme nastavili 5 minútové zahrievacie kolo a 10 minútový beh aplikácie, pričom údaje sa budú vzorkovať každých 5 sekúnd. Všimnite si, že je možné nastaviť aj kritérium pre predčasné ukončenie testu,napríklad dosiahnutím preddefinovaného počtu chybových hlásení a podobne. K neúspechu môže viesť aj nadmerná záťaž. Ak napríklad webovú aplikáciu koncipujeme ako jednu z mnohých ktoré pobežia na hostovanom serveri a takáto aplikácia pri predpokladanom počte prístupov zaťaží procesor servera na 95 percent, dopadne testovanie aplikácie jednoznačne neúspešne a jej vývojárom je potrebné odporučiť štúdium kníh o optimalizácii webových aplikácií.

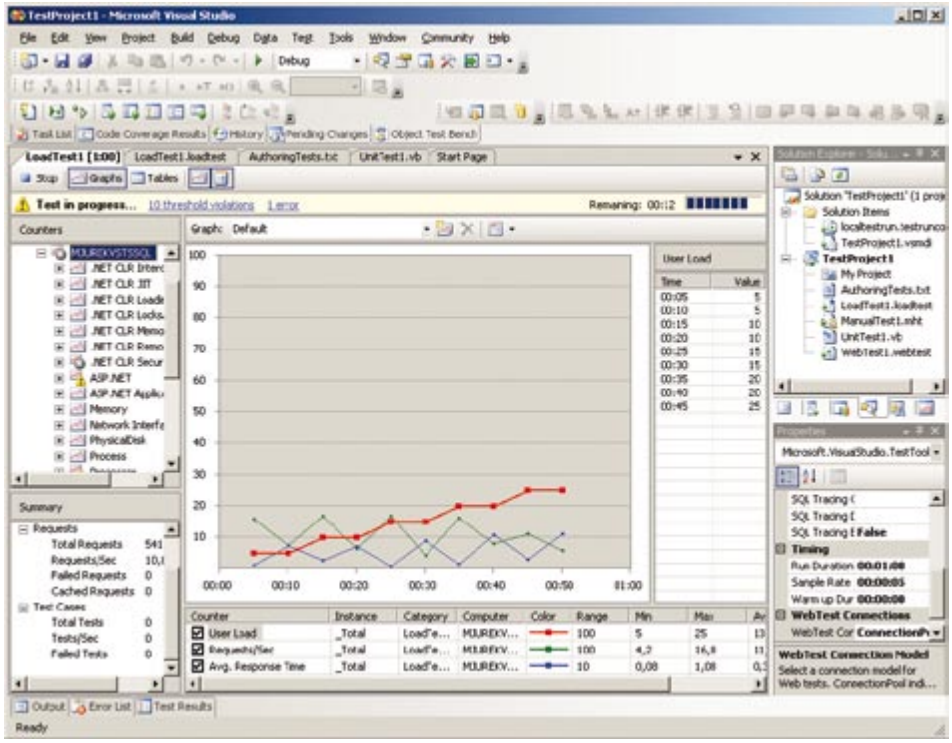


Sumarizácia parametrov pre časovanie testov

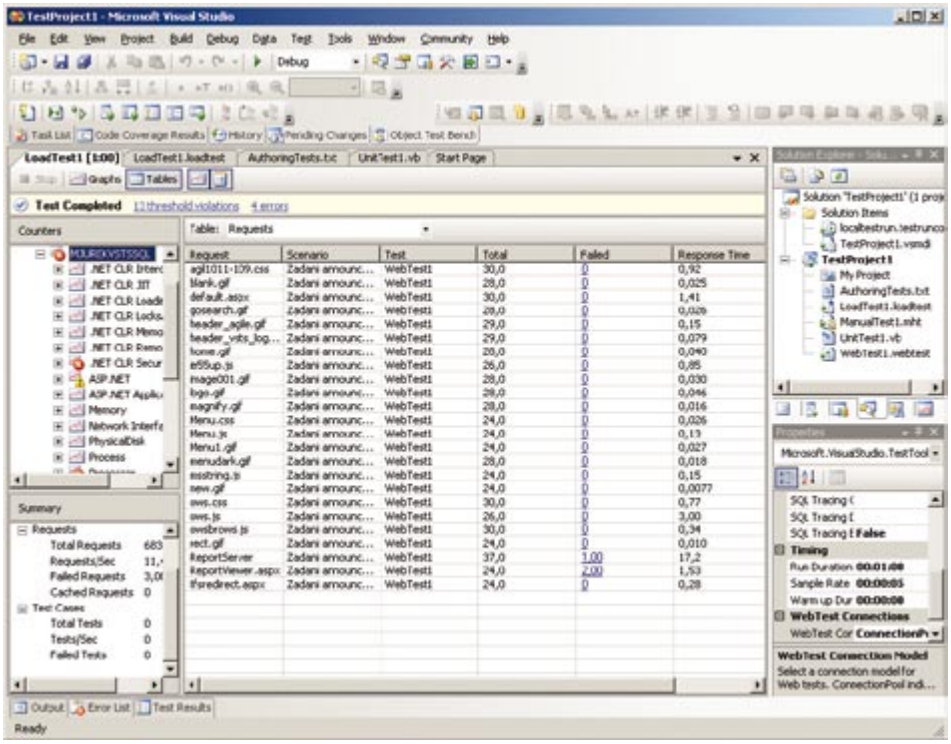


Zobrazenie Load testu vo Visual Studiu 2005

Priebeh testu môžeme sledovať vizuálne pomocou grafov hodnôt jednotlivých čítačov. Po prekročení prednastavených hodnôt sú takéto výsledky testov zobrazené červeným grafom. V strednom najväčšom okne vývojového prostredia sú zobrazené grafy. Vpravo od grafu sú v tabuľkovej forme zobrazené hodnoty pre vybraný graf. Vľavo je okno „Counters" so zoznamom prednastavených čítačov. Body v ktorých sú prekročené prednastavené hodnoty sú na grafe znázornené výstražnými červenými ikonkami.



Zobrazenie priebehu testu



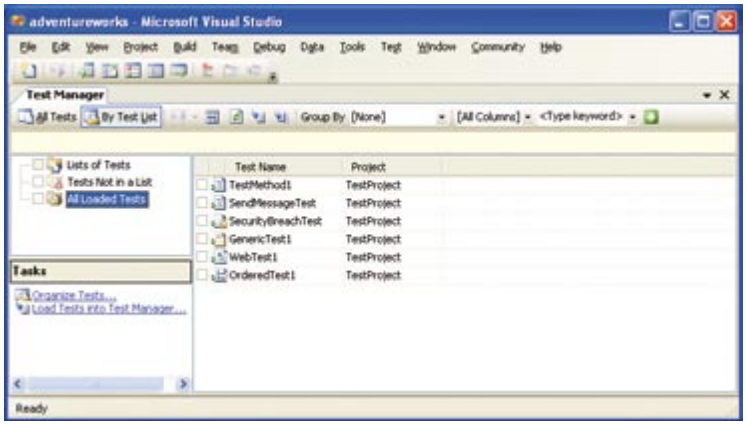
Príklad reportu o priebehu testu

KAPITOLA 8: ĎALŠIE TYPY TESTOV

Ak chceme vykonávať testy v zadanom poradí použijeme „utriedený" (Ordered) test, ktorý slúži na zapúzdrenie iných testov, ktoré spúšťa v špecifikovanom poradí. Ďalším druhom testu je všeobecnýbecný (Generic) test. Principiálne sa jedná o opakované spúšťanie aplikácie s príkazovej riadky s rôznymi parametrami, pričom sa sledujú návratové hodnoty aplikácie. Posledným druhom je manuálny test. Ide v princípe o textový návod pre testerov vypracovaný v textovom editore Word. Testeri vykonávajú podľa tohoto predpisu jednotlivé testy a sami taktiež určujú a zadávajú ako ten ktorý test skončil. Otvorená koncepcia Visual Studia umožňuje doprogramovať vlastné testy

KAPITOLA 9: TESTMANAGER

Vo verzii Visual Studio 2005 for Software Testers môžeme testy komplexne vytvárať a spravovať pomocou nástroja Test Manager. Testy sú usporiadané v hierarchických zoznamoch. Test Manager umožňuje nastavovanie parametrov a konfiguráciu spúšťania testov, sledovať parameter „code coverage“. Testy je možné spúšťať buď individuálne, alebo automaticky na základe vopred vypracovaného zoznamu postupnosti spúšťania testov.



Test Manager

KAPITOLA 10: ZÁVEROM

Pred niekoľkými rokmi bola zo strany Microsoftu vyhlásená iniciatíva Dynamic Systems Initiative (DSI). Pripomeňme si jej ciele na nasledujúcich tri až päť rokov. Mala byť vyvinutá inkrementálna technológia umožňujúca služby automatického zavádzania, efektívnu správu prostriedkov Windows, vyrovnávanie zaťaženia siete a virtuálne servery. Jedným z cieľov je aj zavádzanie technológií pre modelovanie, návrh vývoj šírenie a správu aplikácií. Implementácia tímovej spolupráce, systémového a aplikačného modelovania, analýzy a testovania kódu do Visual Studia 2005 je významným krokom pre naplnenie cieľov tejto iniciatívy. Využívaním týchto nástojov môžeme podstatne zvýšiť efektívnosť a kvalitu procesu návrhu a vývoja, nasadenia a podpory aplikácie naprieč celým jej životným cyklom. Proces vývoja je z hľadiska riadenia kvality rovnocený ostatným podnikovým IT procesom, takže aj v tejto oblasti môžeme uplatniť moderné metodiky a portálové nástroje pre riadenie kvality tímového vývoja a testovania.