DEBRECENI EGYETEM
INFORMATIKA KAR

Visual Studio Team Foundation Server,

Team Build

Témavezető:
Dr. Kuki Attila
Egyetemi Adjunktus

Külső témavezető:
Jagó Jácint
IT Services Hungary Kft.
Head Developer

Készítette:
Sipos Nándor
Programozó Matematikus

DEBRECEN
2009

1. Bevezetés

A rohamosan fejlődő informatika egyre magasabb színvonalú alkalmazásokat, alkalmazásfejlesztési technológiákat követel. Annak érdekében, hogy a minőségi célok elfogadható időn belül megvalósuljanak, fontos, hogy adott probléma megoldásán, adott project fejlesztésében minél több ember részt vehessen. Ugyancsak lényeges szempont lett, hogy a korszerű, akár több száz főből álló alkalmazásfejlesztői csapat tagjai, földrajzilag elszigetelve is dolgozhassanak. Az információáramlás ilyen körülmények között meglehetősen nehézkes. Ezért gondolta a Microsoft, hogy érdemes lenne megoldást találni arra, hogy a Developer Team idejének nagy része ne megbeszéléseken, telefonkonferenciákon, vagy levelezéssel teljen. Megalkotta ezért a Visual Studio Team Systemet (VSTS), egy fejlesztőeszközt, amely a szoftverfejlesztési életciklust kezdetétől – a tervezéstől - egészen a végéig – a tesztelésig - a lehető leghatékonyabban használható.

Dolgozatom céljául ennek a csoport alkalmazásfejlesztői rendszernek a bemutatását tűztem ki, és szeretnék különös hangsúlyt fektetni magára a build folyamatra. Szándékom, egy olyan leírás készítése, amelyből kiderül, hogy a szoftverfejlesztői csapat szereplői hogyan végezhetik munkájukat a legkorszerűbb eszközökkel, úgy, hogy közöttük az együttműködés is minél hatékonyabb legyen. A témaválasztásnál sokat számított, hogy van ilyen rendszerrel tapasztalatom. Munkám során számos termelésben is részt vevő Team Foundation Server 2005 és 2008 rendszer telepítésében, működési kérdés megoldásában, teljesítménnyel, konkrét project-tel kapcsolatos jelenséggel szembesültem, amelyek mind segítettek ennek a technológiának a megismerésében. Úgy gondolom, hogy a rutinos programozók között is előfordulhatnak emberek, akik még nem találkoztak ezzel a programmal. Talán iromány om hatására a Visual Studio Team Foundation Server-t is beveszik a lehetséges opciók közé a Project Manager-ek, amikor csapatuk számára ideális *integrált alkalmazási életciklus management megoldás* (ALM) bevezetésén gondolkodnak.

A Team System ugyanazon az eszközkészleten alapul, amelyet a világ vezető szoftvergyártója használ belsőleg saját szoftverei sikeres tervezéséhez, létrehozásához és telepítéséhez. Olyan alkalmazássá bővíti a Visual Studio-t, amely a teljes csapatot integrálja a termék életciklusának minden fázisában. Ezáltal növeli a project sikerességének megjósolhatóságát, a csapat termelékenységét, hatékonyságát, együttműködését. Mivel szolgáltatásai testreszabhatóak és bővíthetőek, megteremti a fejlesztői környezet bővíthetőségének lehetőségét is.

A VSTS ereje elsősorban olyan csapatoknál érzékelhető, ahol fellelhető a *project manager*, *rendszertervező*, *fejlesző* és *tesztelő* szerepkör. Ezeket a szerepeket, és a hozzájuk kapcsolódó mindennapi feladatokat, elvárásokat szeretném néhány szóval ismertetni. A *project manager*eknek kell áthidalni az űrt az ügyféligények és a fejlesztői csapat munkaterve között. Nekik kell biztosítaniuk a csapat együttműködését és kommunikációját konzisztens és naprakész dokumentációval. Be kell tartatniuk, tartatniuk a határidőket, kezelniük és minimalizálniuk kell a kockázatokat, szükség esetén stratégiai irányváltásokat kell beiktatatniuk, és fenn kell tartaniuk a folyamatok zavartalan hömpölygését a fejlesztői életciklus valamennyi fázisán keresztül. Ezek a feladatok jellegzetesen olyanok, melyeket csak sok független eszközzel, és „kézi" beavatkozással lehet elvégezni. A VSTS leegyszerűsíti a Project Managerek munkáját, mégpedig úgy, hogy a különböző project management eszközöket egyetlen alkalmazáson, a Visual Studio-n belül teszi elérhetővé. Az *architect*ek számára Distributed System Designer-ek (Elosztott Rendszer Tervező) állnak rendelkezésre, amelyekkel átláthatóbbá, rugalmasabbá tehető a Service orientált alkalmazások fejlesztése, telepítése. Lehetővé teszik, hogy míg az alkalmazástervezők ezeket az alkalmazásokat vizualizálják, addig a fejlesztők dolgozni tudjanak a generált kódokkal, mégpedig úgy, hogy a kódváltozatok szinkronban legyenek a designnal. A Distributed System Desinger-ekkel lehet diagramokat létrehozni, melyek az adatközpont logikai felépítését szemléltetik. Ezek segítségével remekül lehet a fejlesztőkkel ismertetni pl. a telepítés célkörnyezetének lényeges adatait. A Code Analysis tool pl. egy éjszakai build check-in policy-jának (beadási szabályok, lsd következő fejezet) részeként, lehetővé teszi, hogy a hibák ki legyenek javítva mielőtt a kód hozzáadódik a forrásfához. Ennek a *fejlesztők* vehetik hasznát

munkájuk során. A teljesítmény monitorozó eszközök segítségével korán bemérhetőek, kiértékelhetőek az esetleges performancia problémák. A múltban a tesztelési és a fejlesztő eszközöket elkülönítve tartották. A *tesztelő*k külön környezetet, és test scripteket használtak, melyeket ők írtak, és a forráskód kezelőtől elkülönítve tároltak. A VSTS egy rakás teszt eszközt is integrál az IDE-n belül. A Web tesztek, load tesztek, manuális tesztek és unit tesztek (egység tesztek) mind saját varázslóval és szerkesztőablakkal rendelkeznek. A tester-ek szerkeszthetik, futtathatják és ellenőrizhetik a különböző munkaegységekhez kapcsolódó teszteket a VS-en belül. Egy sikertelen teszt esetén mindössze egy jobb-klikk elég ahhoz, hogy létrejöjjön egy bug, és hivatkozás is készíthető a problémás kód részletre. Így a probléma könnyen reprodukálható. Rendelkezésre áll egy bővíthető infrastruktúra arra is, hogy a tester-ek, developer-ek és a partnerek létre tudják hozni saját teszt típusaikat, szerkesztőiket, és visszajátszó mechanizmusukat, ami a Visual Studio reporting és analysis rendszerrel való együttműködésének eredménye. Az említett 4 szerepkör között nincsenek szilárd határok, bővíthetőek, sőt a csapat felépítésétől és a project jellegétől függően egy személy akár több szerepet is betölthet.

Miután néhányat említettem azok közül, hogy „melyek" azok a funkciók, amivel a VSTS segíti a szereplők munkáját, dolgozatom során jónéhány technikai „hogyan" kérdésre is igyekszem választ adni. A teljesség igénye nélkül igyekszem nagyobb hangsúlyt fektetni azokra a funkciókra és technológiai megoldásokra, amelyek véleményem szerint a VSTS magját, lényegi részét képezik, s hogy mik azok az ötletek, amelyeket a világ vezető softvergyártói is alkalmaznak eredményességük receptjeként.

2. A Team Foundation Server

2.1 Alapfogalmak

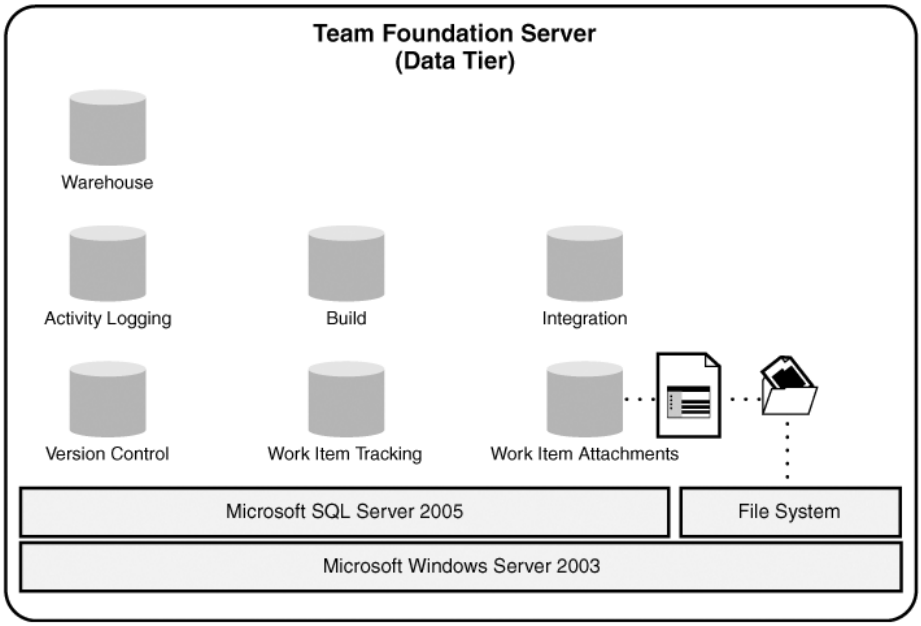
Ahhoz, hogy mélyebben elkezdhessük taglalni ennek a fejlesztői csoportot támogató eszköznek a felépítését, elengedhetetlennek érzem, hogy néhány alapvető fogalmat tisztán lássunk. Ennek a fejezetnek ezt tűztem ki céljául. Aki már találkozott a VSTS-sel és megismerkedett alapfogalmaival, most néhány soron keresztül nem fog újat hallani.

A TFS egy többrétegű architektúra[1]: **Adatbázisrétegből** (Data Tier, DT), **Alkalmazásrétegből** (Application Tier, AT), és **Kliensrétegből** (Team Explorer, vagy maga a Visual Studio) áll. Felépítése tehát hasonló ahhoz, amit nagyvállalatok szoktak létrehozni saját üzleti alkalmazásaikhoz, és a felhasznált software-k is a leggyakoribbak: ASP.Net, WSS és Microsoft SQL Server.

Adatbázisréteg (Data Tier): a TFS adatbázisrétegét az Microsoft SQL Server hosztolja. Ideális esetben ez egy dedikált server, ami csak ezt az egy feladatot látja el. A TFS az SQL Server adatbázismotor és analitiai, elemző szolgáltatásait használja. A DT-be az alábbi adatbázisok kerülnek a telepítés során (TFS 2008 esetében is):

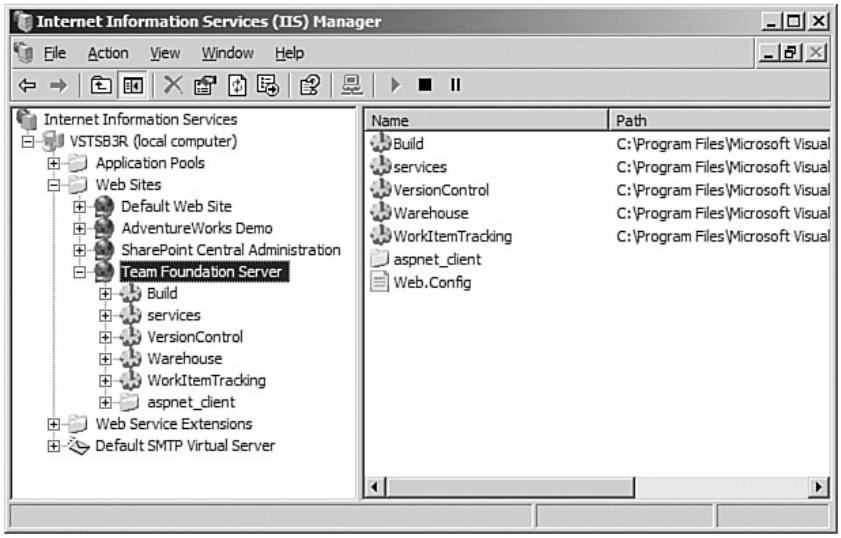
- *TFSIntegration* – TFS központi szolgáltatások (project metaadatok, figyelmeztetések, security group-ok, stb)
- *TFSWarehouse* – TFS adattárház (analízis és report tároló)
- *TFSWorkItemTracking* – itt tárolódnak a work item-ek (arról, hogy mik azok a work item-ek, a későbbiek során ejtek szót)
- *TFSWorkItemTrackingAttachments* – ennek külön táblája van (Attachments), ez egy mutatógyűjtemény a work item attachment file-jaira.

- *TFSVersionControl* – verziókezelési adatbázis (lsd. Később: Verziók)
- *TFSActivityLogging* – activity log adatbázis. A TFS rendelkezik egy beépített eszközzel arra, hogy logoljon minden server parancsot a DT adatbázisai felé. Ezek mind web service hívások.
- *TFSBuild* – build adatbázis, ami a build-ekkel kapcsolatos adatokat tartalmazza



Alkalmazásréteg (Application Tier, AT): Az AT ASP.net webservice-ekből, valamint a csapatportált támogató WSS-ből áll. (Megjegyzés: mivel mind a TFS webservice-k, mind a WSS ugyanarra az IIS telepítésre támaszkodnak, különböző portszámot kell hozzájuk beállítani.)

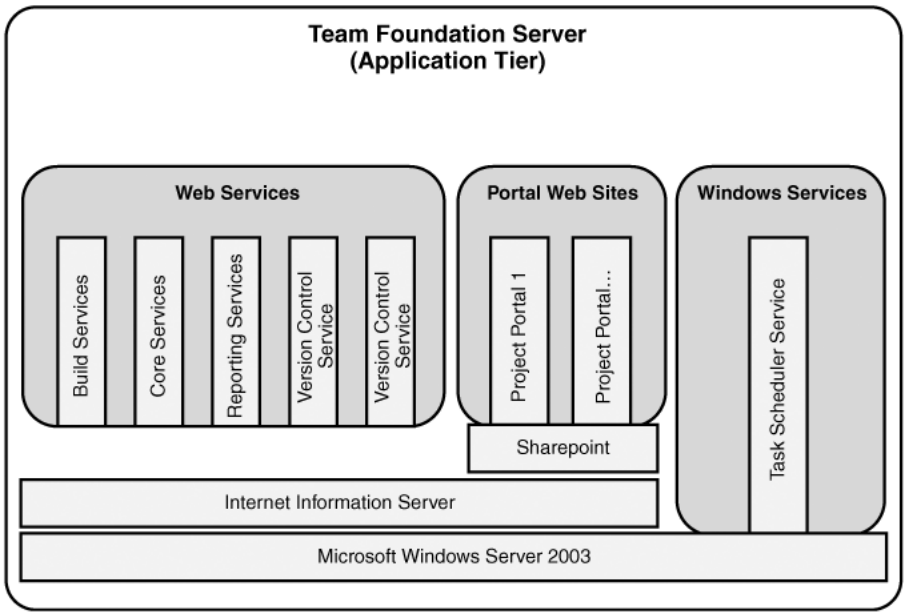
A *web service*-k az AT-n a TFS API-re épülnek, és ezek biztosítják a TFS funkcionalitásait. A service-k virtual directory-kban találhatóak az IIS Manager Team Foundation Server website gyökere alatt:



A webservice-ket nem közvetlen használatra tervezték. A szoftverfejlesztő készlet (SDK, Software Develpment Kit) tartalmaz további információkat erről. Ettől függetlenül lehetséges a közvetlen kapcsolat, ha egy webhivatkozás megadásával a megfelelő webservice-re.

A *WSS*-sel (*Windows SharePoint Services*) könnyen és gyorsan létrehozható a Team Project portal környezete bármilyen méretű team számára, méghozzá anélkül, hogy kódot kellene írnunk. A WSS portálok előnye továbbá, hogy képesek az összes szükséges művelet elvégzésére, amelyre szükségünk lehet. Pl: feltöltés, letöltés, biztonság, menük, Check-in, Verziókezelés, stb.

A következő ábra az AT felépítését szemlélteti:



Kliensréteg (Client Tier, CT): a TFS kliensrétegét a következő alkalmazások képezhetik:

- Visual Studio 2005 vagy 2008 – fejlesztői környezet
- Microsoft Office Excel
- Microsoft Office Project
- Microsoft Internet Explorer
- Team Explorer – VS-be épülő modul, vagy különálló eszköz a Team System project-ek létrehozására és kezelésére.

Ezek az alkalmazások a TFS-el a webservice-eken keresztül kommunikálnak.

Munkaelem (Work Item, WI): Önálló munkaegység, melyet el kell végezni. A munkaelem kifejezés meghatározhat bármilyen tevékenységhez köthető információt, amely része a szoftverfejlesztési életciklusnak. Olyan követelmények, feladatok, hibák, stb., melyeknek tulajdonságaik vannak, úgymint címük, dátumuk, állapotuk, és a csapattag neve, akihez rendeltük. Ezek az elemek a szoftverfolyamatunk alapjai lehetnek, és ahogy haladunk az időben a project-ünk fázisain keresztül, úgy fejlődnek ezek is velük. A workitem-eket más objektumokhoz tudjuk kapcsolni. Ha verziókezelőbe beadott kódhalmazhoz rendeljük, akkor a fejlesztő, vagy projectvezető nyomon tudja követni, mely változtatások tartoznak melyik követelményekhez, illetve hibákhoz.

Verziókezelés (Version Control): Olyan több verzióval rendelkező adatok kezelése, melyeken egyidejűleg nem csak egy ember dolgozhat. Az egyes változtatásokat verziószámokkal vagy verzióbetűkkel követik nyomon. A Team Foundation Verziókezelő (Team Foundation Version Control, TFVC) bevezeti a változáskészlet, *changeset* fogalmát. Más verziókezelő termékeknél az egyes fájloknak, amelyeken dolgozunk, nincs hivatkozásuk vagy környezetük: csak külön fájlok, melyeket egy időben adunk be. A TFVC-ben a changeset-ek összetartozó fájlmódosításokat írnak le. Ezáltal a könyvelés és a munkafolyamat jobb, az administzráció pedig egyszerűbb. Minden chageset egy egyedi azonosítót kap a nyomon követéshez, és a jelentésekhez.

Check-out (kivétel): Lokális másolat készítése egy verziókezelt file-ról. Ilyenkor legtöbbször a legfrissebb verziót kapja a user, de van lehetőség konkrét (régebbi) verzió kikérésére is.

Check-in (beadás): Az a művelet, amikor a lokális példányváltoztatások jóváíródnak a serveren tárolt változatban. Ez történhet egyszerű másolás vagy összefésülés (merging, lásd később) eredményeként.

Conflict: Konfliktusról akkor beszélünk, ha többen akarnak megváltoztatni egyszerre egy adatot, és a rendszer képtelen a változásokat összeépíteni. Ilyenkor a felhasználónak kell feloldania a konfliktust. Ezt úgy teheti meg, hogy kiválasztja az egyik változtatást és csak azt juttatja érvényre.

Change (változtatás): Verziókezelt adaton alkalmazott változtatás.

ChangeList (változtatás lista): egy check-in alatt bevitt változtatások listája.

Branching (elágaztatás): Verziókezelt adatok egy részhalmaza elágazhat, így több aktuális változattal is rendelkezhetnek. A branching az a művelet, amikor kimásoljuk az aktuális forrás fából a kód-ot, és beillesztjük a fa újonnan létrehozott ágába. Hogy ezt könnyebben megértsük, vegyünk egy példát a file rendszer-ből. Elágaztatáskor vesszük az aktuális kód mappáját (forrás) és létrehozunk egy új mappát beillesztve a forrás mappa tartalmát. Gondolhatunk tehát az elágaztatásra úgy is, mint egy file rendszer szintű másolás művelet, melynek során vesszük az egyik helyen lévő file-okat, és átmásoljuk őket egy másik helyre. Ezekután a fejlesztők párhuzamosan folytathatják a munkát mindkét mappa tartalmán.

Mergeing (összefésülés): Az összefésülés az elágaztatás ellentett művelete. Egy elem elágaztatása helyett a forrás fában, két elemet alakítunk eggyé.

Source Control(adattárház): a Team Foundation Server (Team Foundation Verson Control, TFVC) verziókezelőjét Source Control-nak, vagy Repository-nak (adattárház) nevezzük. Eltérően a Microsoft korábbi verziókezelőjétől (Visual SourceSafe), amely még file rendszer alapú tárolási mechanizmussal működött, a TFS Source Control minden forrásfile-t, változtatást, az aktuális check-out-okat, stb. SQL adatbázisban tárolja. A Source Control-nak köszönhetően áll a felhasználó rendelkezésére a többszörös, vagy szimultán check-out, konfliktuskezelés, shelving, unshelving, a biztonsági szintek állítási lehetősége a forrásfa minden szintjén,

Shelving: Forráskód időleges elhelyezése a Version Control server-en, tényleges check-in nélkül.

Workspace: kliens oldali másolata a source control serveren lévő file-oknak és mappáknak. Mikor hozzáadunk, szerkesztünk, törlünk, mozgatunk, átnevezünk forráskezelt egyedeket, a változtatásaink késleltetődnek, vagy megjelölődnek pending changes-ként (felfüggesztett változtatások) a workspace-ben. A workspace egy elkülönített hely, ahol írhatjuk, tesztetlhetjük a kódot, anélkül, hogy aggódnunk kellene a már beadott forráskód stabilitása miatt, vagy amiatt, hogy a többi fejlesztő változtatásai hogyan fogják befolyásolni a pillanatnyi munkánkat. A változtatásaink elkülönítődnek a workspace-ben mindaddig, amíg be nem adjuk őket a source control serverre. A Get Latest parancs való a workspace-ünk synchronizálására a legfrissebben beadott változtatásokkal a serveren.

Build (összeállítás): Forrás kódból, a független, végrehajtható, és számítógépen futtatható szoftvertermékké való konvertálás folyamatát, és a folyamat végeredményét Build-nek nevezzük. Ez a folyamat egyszerűbb programoknál mindössze egyetlen forrásfile lefordításából áll, összetettebb szoftvereknél viszont a forráskód több file-ból is felépülhet, és különböző módon kombinálható annak érdekében, hogy több verzió jöjjön létre. A Build process-t általában egy build eszköz vezérel, amely lefordítja, és összekapcsolja a megfelelő sorrendben a file-okat. Ha a forráskód egy bizonyos file-ban nem változott meg, akkor nem szükséges újrafordítani, ami a build idő optimalizálása szempontjából lényeges.

MSBuild: A Microsoft Build platform-ja, amit jellemzően a Visual Studio-val használnak. Az MSBuild 2.0 a .NET Framework 2.0 része, és a Visual Studio 2005-tel működik együtt. A 3.5 verziót a .NET Framework 3.5 tartalmazza, a Visual Studio 2008-hoz adták ki, és lehetőség van vele 2.0, 3.0 és 3.5 .NET Verziók által támogatott project-ek build-elésére. Ezt nevezzük multi-targeting-nek. Mivel az MSBuild a .NET Framework része, lehetőség van Visual Studio project-ek, solution-ok build-elésére anélkül, hogy a Visual Studio IDE telepítve lenne a gépünkön. Az MSBuild project file-okkal dolgozik. Ezek XML file formátumúak, kiterjeszthetők, és segítségükkel meghatározhatjuk mely egyedeket kell build-elni, és azt is, hogy hogyan kell ezt tenni különböző platformok, konfigurációk esetén. Újrafelhasználható

build szabályokat készíthetünk, melyeket különböző file-okban tárolhatunk, lehetővé téve, hogy a termék különböző project-jeit konzisztensebben build-elhessük.

2.2 Visual Source Safe

Minden fejlesztési projecthez, ami egyszerű demókódnál többet jelent fontos, hogy tartozzon hozzá verziókezelő. A Microsoft berkein belül ezt a célt kezdetek-kezdetén a Visual Source Safe (VSS) szolgálta. Annak ellenére, hogy a VSS hasznos eszköz, sok fejlesztő szeretett volna több fejlett funkciót, lehetőséget használni. A Microsoft ezért tűzte ki céljául, hogy egy olyan fejlett, teljeskörű, vállalati szintű verziókezelő rendszert alkosson meg, amely az összetett fejlesztési feladatok többségének kezelésére alkalmas, továbbá a VSS-nél sokkal megbízhatóbban kezeli az elágaztatás és összefésülés műveleteit. 2006-ban ezért piacra dobta a Visual Studio 2005 Team Foundation Server-t (TFS), amelyben számos új funkció található. Írásom most következő részében szeretnék kitérni a VSS és a TFS architektúrális és funkcionális különbségeire[3], melyeket érdemes fontolóra venni, amikor döntésre kerül a sor azt illetően, hogy melyik technológia lenne ideálisabb választás fejlesztői csapatunk számára.

2.2.1 A TFS szerkezeti előnyei: a VSS a Source Safe explorerével és a Visual Studio plug-in-jével egyaránt a Visual SourceSafe adatbázisba ír és onnan olvas, ami lényegében egy file gyűjtemény, általában egy megosztott hálózati mappában. Ezzel szemben a TFS esetén a source control rendszer .NET Web service-t használ, hogy hozzáférjen az adatokhoz, melyeket egy SQL Server adatbázisban tárol. Emiatt a TFS source control architektúrája jobb teljesítményt és nagyobb megbízhatóságot garantál.

Biztonság és project jogosultságok: a VSS esetében felhasználói jogosultságok és kinevezéseket (melyeket a VSS administrator programmal lehet beállítani), függetlenek a Windows Sharing permissions-től (megosztási engedélyek), melyeket a VSS megosztott hálózati mappára állítottunk be. Be lehet állítani a jogokat és kinevezéseket konkrét VSS projectekhez vagy független VSS felhasználókhoz, de minden VSS user-nek ugyanolyan engedélyekkel kell rendelkeznie a VSS adatbázis mappa tekintetében. Ennél fogva minden VSS user hozzáférhet a megosztott folder-hez, és teljes jogosultsággal rendelkezhet a VSS adatok fölött, függetlenül a project szintű jogaitól, melyet a VSS admin programban specifikáltunk. A TFS esetén a user specifikus operatív engedélyek és a project szintű engedélyek a Windows user account-hoz kötődnek. A user autentikációt biztonságosan végzi az Internet Information Server (IIS). Külön SQL Server adatbázis hozzáférésre nincs szüksége az egyes user-eknek, akik source control műveleteket hajtanak végre. Ezek eredményeként a TFS security infrastruktúrája könnyebben adminisztrálható, robusztusabb és biztonságosabb.

Megbízhatóság: mivel a VSS-nek nincs server komponense, az adatszállítással járó műveletek nem tranzakcionálisak, azaz nincs lehetőség roll-back-re ha probléma történik. Azon ritka esetekben, amikor megszakad a hálózati kapcsolat egy írási művelet közepén, az érintett file-ok integritása veszélybe kerül, és információ vész el. A TFS egy kliens-szerveralkalmazás, melyben a műveletek adatbázisban történnek stored procedures (tárolt eljárások) útján, melyek nincsenek kitéve hálózati kapcsolati hibáknak. Bizonyos műveletek tranzakcionálisan hajtódnak végre, minél fogva hiba esetén lehetőség van roll-back-re. Ez az arhitektúra biztosítja, hogy a forrásfile-ok ne korruptálódhassanak. Biztosak lehetünk abban is, hogy azok a file csoportok, melyek összefüggő change-eket tartalmaznak ugyanabban az időben lesznek a source control serverre commit-álva egy megfelelően ellenőrzött changeset-ként.

Skálázhatóság: TFS támogathat akár 2000 user-t is, míg a VSS 20, vagy kevesebb létszámú team-eknek ajánlott. A TFS annyi adatot tárolhat, amennyit az SQL Server elbír (terabyte-ok…) és amennyit a hardware környezet megengedhet, ugyanakkor a javasolt méretkorlát a VSS adatbázisa esetén 4 GB.

2.2.2 A TFS Funkcionális előnyei:

Changeset-ek: Koncepcióját tekintve a VSS adatbázis és a TFS soruce control server belső strukturája hasonló. Mind az adatbázisok, mind a serverek hierarchikusan szerveződnek. Mappák tartalmaznak file-okat. A file-oknak verziói vannak, melyek szám és létrehozás ideje/dátuma alapján azonosíthatók. A TFS-nek viszont van egy előnyös koncepciója, ami még nem létezik a VSS-ben. Ez pedig nem más, mint a changeset. A changeset egy logikai tároló, amiben a TFS minden check-in művelethez kapcsolódó dolgot tárol: file és mappamódosítások, linkek a kapcsolódó work item-ekre, check-in kommentek és más információk, mint pl. hogy ki végezte a változtatást.

Check-out és check-in különbségek: VSS-ben csak akkor kell explicit check-out-ot és check-in-t végezni, ha file-t szerkesztünk. TFS esetén, minden cselekvésnél szükség van explicit check-out és check-in-re.

További eszközök, melyek különbözően működnek: VSS-től eltérően a TFS nem hajt végre GET műveletet, mikor check-out-olunk egy file-t. Egyszerre több user is check-out-olhat és változtathatja ugyanazt az item-et. VSS esetén a check-out-ok kizárólagosak. TFS-nél viszont lehetőség van a file lock-olásra, hogy más user-ek ne check-out-oljanak vagy check-in-eljenek változtatásokat.

Elágaztatás és összefésülés: a VSS csak nagyon alap szinten támogatja a branch és merge műveleteket, mert nem tárolja a merge history-t a két file, vagy folder branch között. Ezzel szemben a TFS source control esetén létezik merge history.

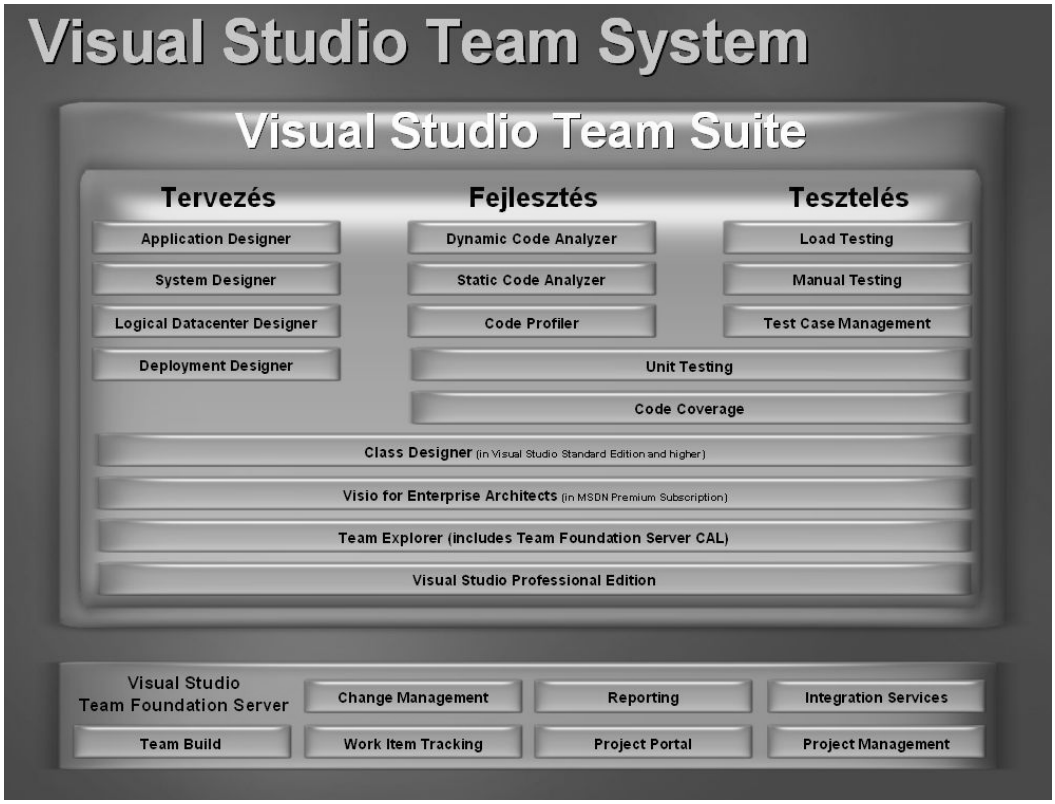
Checkout művelet és a Get Latest: mikor végrehajtunk egy Get operációt, hogy feltöltsük a workspace-ünket file-okkal, egy konzistens pillanatfelvételt készítünk a source control-ról. A forrás konfiguráció kijelöl egy időpontot, amikor felvétel készül a TFS source control azon file-jairól, melyeket összetartozónak vél, és így együtt build-elhetőek és tesztelhetőek. Mikor egy fejlesztő dolgozik a workspace-ben, védett a többi fejlesztő által végzett változtatásoktól. Ő irányítja, mikor akarja elfogadni a mások általi változtatásokat, úgy, hogy végrehajtja a Get műveletet. Mikor ez történik, frissítve lesz az egész forrás konfiguráció és nem csak egy-két file. Egy file változtatása függ más megfelelő file, file-ok változtatásain, és emiatt meg kell győződni róla, hogy konzisztens a forrás snapshot, amit build-elni és tesztelni akarunk. Emiatt nem hajtódik végre CheckOut művletkor Get Latest a CheckOut alatt levő file-okon. A kivétel alatt lévő file update-elése ezt a konzisztens snapshot filozófiát tönkre tenné és olyan forrás konfigurációt hozna létre, amely nem build-elhető, tesztelhető. Egy lehetséges megoldásként a TFS kényszeríti a user-t, hogy végrehajtson Get Latest műveletet még mielőtt beadja a változtatásait, így ha a beadáskor nem a legaktuálisabb másolatunk van, a TFS a konfliktus kezelési párbeszédablak feldobásával figyelmeztet.

Ezek tehát az alapvető különbségek a VSS és a TFS között, melyek felsorolása után azt hiszem már könnyebben eldönthető, hogy a VSS, avagy a TFS a jobb választás a team-ünk számára. 2006 és a TFS első verziójának megjelenését 2008-ban a Visual Studio 2008 Team Foundation Server megjelenése követte, amelyben számos olyan új dologot változtattak, vagy oldottak meg, amely az első verzióból az "ínyenceknek" esetleg hiányozhatott. Ezek közül csak néhányat említenék felsorolásképpen[2]:

- SharePoint 2007 támogatás (és immáron tetszőleges porton keresztül is)
- Reporting Services támogatás (másik szerveren, és extra portokon is)
- Nevesített SQL Server támogatás (így már egy szerveren akár több TFS is futhat)
- Windows 2008 támogatás
- SQL 2008 támogatás
- Upgrade támogatás TFS 2005-ről
- Performancia optimalizálás (akár 30.000 user is lehet)
- Egyszerűsített telepítés (nincs külön DT telepítés, szükségtelen annyi speciális user a telepítéshez, és nem kell mindnek admin-nak lennie sem)
- Végleges törlés
- Certificate alapú azonosítás
- Stb.

2.3 A Visual Studio Team System

A bevezető során már felszínesen kifejtettem, hogy a VSTS a négy szerep mindegyikét külön funkciókkal, lehetőségekkel támogatja. Annak érdekében, hogy a különböző szerepek esetlegesen mások számára szükségtelen funkciói ne bonyolítsák túl az alkalmazást, mindegyik szerephez létezik külün Visual Studio edition(kiadás). Most ezen kiadások sajátosságaira szeretnék kitérni. A következő táblálzat rendszerezi az egyes eszközöket. A különböző kiadások különböző feladatkörre specifikált modulokat, opciókat tartalmaznak[6]. Ezeket a következő ábra rendszerezi:



2.3.1. Team Explorer

A Project Managerek számára a legfontosabb a *Team Explorer* (TE) lesz, amely egy ún. add-in (beépülő modul) a Visual Studio-hoz. Segítségével tud kapcsolódni a TF serverhez. Ha nincs szüksége Visual Studio-ra, akkor dönthet úgy, hogy csak TE-t telepít gépére, mert erre is lehetőség van, és a Team Explorer is tartalmazni fogja a munkájához szükséges legfőbb funkciókat. Létre tud új Team Project-et hozni, kiválaszthatja a process template-et (telepítéskor MSF for Agile, és MSF for CMMI Software Development process template-k vannak, de bármilyen egyéni minta létrehozható). Ki tudja osztani jogosultságokat a TFS-hez, hozzáférhet a Work Item-ekhez (újat hozhat létre, szerkeztheti azokat, követheti állapotukat), dolgozhat a Project SharePoint portálján (ez szintén magától létrejön a Team project létrehozásakor), az összes dokumentum elérhető számára, ami a Team Project-hez kapcsolódik. Szintén fontos a lifecycle során, hogy a project manager jelentéseket tudjon lekérni. Erre a TFS a Reporting Services-el (RS) nyújt megoldást. Mivel az RS böngésző alapú, nem kell külön eszközt telepíteni a jelentések kiíratásához, kinyomtatásához. Az SQL Server komponenseként települ a TFS DT-re. Mivel a legtöbb project manager az MS Office Excell-t használja reportjai elemzéséhez, fontos megemlíteni, hogy a VSTS használatakor is lehetőségük lesz ad-hoc lekérdezéseket, jelentéseket készíteni Excell-el, mivel a táblázatkezelővel is lehet a TFS adatbázisához kapcsolódni, és adatokat megjeleníteni Pivot Tábla formájában. A lekért adatok megoszthatóak a csapattagokkal a project SharePoint Portálja segítségével.

2.3.2. Visual Studio 2008 Team System Architecture Edition

A csapatunk Architectjei[7] számára a Distributed System Designer (Elosztott Rendszer Tervező) lesz a legnagyobb érték, melyet a *Visual Studio 2008 Team System Architecture Edition* tartalmaz. A Distributed System Designer-ek az első hordozói a Dinamikus Rendszer Kezdeményezésnek (DSI, Dynamic System Initiative), amelynek szándéka, hogy megkönnyítse az elosztott rendszerek tervezését, bevetését, működtetését A Distributed System Designer-eknek köszönhetően a software architect-ek, infrastruktura architect-ek és fejlesztők növelhetik a termék sikerének előreláthatóságát. Mögöttes metamodel-ként a Rendszer Definíciós Modelt (SDM, System Definition Modell) használják. A Distributed System Designer jónéhány grafikus tervezőből áll, amelyek a következő célokkat szolgálják:

- dokumentálni, visualizálni és közzétenni az alkalmazás terveket, adatközpont run-time környezeteket.
- elosztott alkalmazás rendszereket tervezni
- alkalmazás rendszereket értékelni ki a cél adatközpontban
- létrehozni és összeegyeztetni az alkalmazás terveket a forráskódokkal

Hogy ezeket a célokat még magasabb színvonalon elérhessék az architectek és fejlesztők, a következő eszközök is rendelkezésükre állnak:

- Application Designer (alkalmazástervező): a fejlesztői környezetben megtervezi, vizualizálja, konfigurálja és implementálja az alkalmazásokat
- System Designer (rendszertervező): megtervezi az alkalmazásokat tartalmazó-, és egyéb rendszereket.
- Web Service Details window (Webszolgáltatások részletei): megtervezi és átnézi a Web Service műveleteket.
- Setting and Constraints Editor (Beállítás és megszorítás kezelő): az alkalmazás és logikai szerver beállításokat és constraint-eket kezeli.
- Logical Datacenter Designer (Logikai Adatközpont tervező): logikai adatközpontként ábrázolja a futtató környezeteket, és segít a rendszer skálázhatóságát felmérni
- Deployment Designer (telepítés tervező): definiálja és kiértékeli a alkalmazások,

alkalmazásrendszerek telepítés definícióit.

2.3.3. Visual Studio Team System 2008 Development Edition

A *Visual Studio Team System 2008 Development Edition* olyan haladó eszközkészlettel látja el a fejlesztőket, amely segít felismerni a rossz minőségű, nem elég biztonságos, vagy nem hatékony kódot. A programozók rendelkezésére bocsátja a legjobban bevált praktikákat, továbbá az automatikus Egység Tesztelés (Unit Test) lehetőségét. A Unit Test a minőség biztosításának legjobb módja. Olyan teszteket kell írni, amelyek ellenőrzik az adatokat és algoritmusokat, valamint biztosítják, hogy a kezdeti hibák nem térnek vissza később. Ennek érdekében 4 fajta Unit Test van:

- A pozitív egységtesztek valósághűen lefuttatják a kódot és igazolják a helyes eredményt
- A negatív egységtesztek szándékosan helytelenül futtatják a kódot és igazolják a megfelelő teljesítményt és hibakezelést.
- A stress test-ek vagy feszültség tesztek a kód teljesítményének határait próbálják felfedni és ismertetik azokat.
- A hibabeszúró (Fault Injection) tesztek a hibakezelés rendellenességeit fedik fel.

A bug-ok korai felfedezésének legegyszerűbb módja, ha a fejlesztők növelik a figyelmeztetések szintjét a szerkesztőben és a kódelemzési eszközökben. Ha megfelelően használják a workitem-eket, akkor pedig nem maradhat befejezetlen kód.

A teljesítmény vagy performance eszközök segítségével mérhetőek, kiértékelhetőek a teljesítmény problémák. Ezek az eszközök mind a Visual Studio-ba integráltan találhatóak meg.

2.3.4. Visual Studio Team System 2008 Test Edition

A *Visual Studio Team System 2008 Test Edition* teszt eszközök garnitúráját tartalmazza, melyek nem csak a saját teszt keretrendszerükben, hanem más nagyobb szoftver életciklus eszközök szerkezetén belül is. Ez a kiadás a teszterek számára megkönnyíti, hogy létrehozzanak, vezessenek, szerkesszenek, futtassanak, tároljanak teszteket. Több teszttípus, mint pl. Web, load (terhelés) és kézi teszt található benne. Lehetőség van létrehozni egy olyan több számítógépekből álló group-ot, amivel terheléses tesztet lehet szimulálni. A csoport egy vezérlőből, és több agent-ből (ügynök) áll. A csoportot együttesen köteléknek nevezzük. Egy agent annak a köteléknek a része, amely teszteket futtat, és szimulált terhelést generál. A controller, vagy vezérlő pedig annak, amely kötelék vezérli az ügynököket és begyűjti a tesztek eredményeit.

Amikor a tesztek futnak, az eredményeket automatikusan generálják, lementik a lemezre, és megjelenítődnek a Test Results (Teszt Eredmények) ablakban. Töb teszt típusért még mélyebb eredményeket is lehet lekérni. Ha a csapat arra használ egy TFS Team Project-et, hogy segítsen a munkát vezetni, akkor szintén lehet teszteredményeket kiadni. Publikáció után össze lehet olvasztani, és fel lehet használni ezeket a jelentésekben.

3. A Team Foundation Build

A Team Foundation Build (TFB) a Microsoft Visual Studio Team System fejlesztői környezet része. Ahogy a projectek életciklusuk során fejlődnek, előbb vagy utóbb elérkezik az idő, amikor le kell őket fordítani (compile), distribute-olni, és kezdődhet a tesztelés. Ezt a folyamatot együttesen Build-nek nevezzük. A TFS esetén azonban a Builld jóval összetettebb folyamat, mint ahogyan azt az MSBuild-nél tárgyaltuk, és a Project Team tagjai számára rengeteg manuális többletmunkát jelent. Ez a folyamat a következő problémákat rejti:

- A mindenre kiterjedő és a project környezetbe integrált build eszközök nélkül előreláthatatlan és megismételhetetlen, ad hoc build process-eket eredményezhetnek.
- A build process-t figyelemmel követni, tisztán látni a pillanatnyi állapotát szintén nem egyszerű dolog.
- Általában nincsenek módszerek arra, hogy hogyan oldjuk meg a build során esetlegesen felmerülő akadályokat.

A Team Foundation Build a build process számos aspektusát automatizálja, és jelentős eszközkészletet biztosít a folyamat során felmerülő problémák megoldására és a létrejövő build-ek elemzésére. A következők során a Team Foundation Build tool lehetőségeiről, a build rendszer alapelveiről, a build eszközei és a Visual Studio közötti interakcióról lesz szó, továbbá arról, hogyan is épül fel egy tipikus build process a Visual Studio Team System-en belül.

3.1 A Team Foundation Build áttekintése

Számos fejlesztő cég külön build labort hoz létre azzal a céllal, hogy ott a buildeljék publikus és privát software release-eiket. Egy build labor, olyan hardware és software erőforrások együttese, amelyek összegyűjtik a Team Project forráskód file-jait a Build Serveren, majd compile-t hajtanak végre a rendszer legutóbbi változtatásain. Gyakori szokás szerint ezt a build-et később becsomagolják, és egy olyan helyre másolják, amely a Team minden tagja számára elérhető. A minőségbiztosításért felelős szervek ezután már szabadon futtathatják rajta teszt szoftvereiket, és vizsgálhatják, milyen jellegű változtatásokat kell még a fejlesztői csapatnak a project-en végrehajtani.

A Team Foundation Build nem titkolt célja, hogy egy ehhez hasonló Build Labort biztosítson egyetlen dobozba zárva. Lényegében arra törekszik, hogy megszüntesse a build folyamattal járó manuális munka nagy részét, és információt szolgáltasson az aktuális Build állapotára vonatkozóan a fejlesztői Team tagjai számára. A Team Foundation Build platform előnyei:

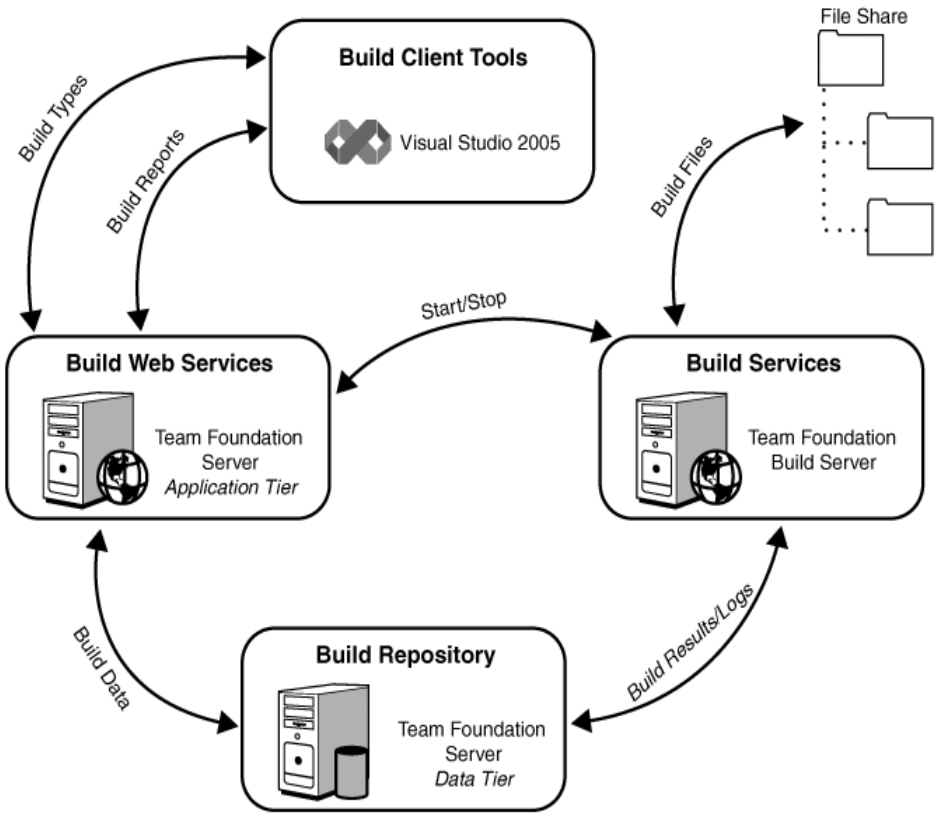
- Lehetővé teszi, hogy a Visual Studio-n belül, gyorsan és könnyen jöjjön létre és kezdődjön el a Build process.
- Eszközöket biztosít a Project Team tagjai számára, amelyekkel a Build státuszát ellenőrizhetik.
- Build-ről build-re összehasonlítható vele akár történelmileg is, hogy hogyan alakulnak az egyes build-ek a tesztelés, fejlesztés során.

3.2. A Team Foundation Build felépítése

A Team Foundation Build szolgáltatáscsomagja 4 Visual Studio-n belüli alrendszerből épül fel. Mindegyik fontos szerepet játszik a build folyamatban[4]:

- A Team Build Client (lényegében maga a Visual Studio) definiálja a build-et a Team Foundation Source Control system-ben, majd el is indítja.
- A Team Foundation Server Application Tier (alkalmazás szerver) figyeli a Team Build Kliens felől érkező build kéréseket, majd továbbítja azokat a Build Server felé.
- A Build Server végzi a munka oroszlán részét: futtatja a Build Service-t, ami vizsgálja a Build specifikációit, begyűjti a szükséges file-okat a Source Control rendszerből, compile-t (fordítást) hajt verge a forrásfile-okon, teszteket futtat, majd az eredményeket publikálja. A Build befejeződése után az értesítések kiküldhetőek a csapatnak, továbbá a Build Server egy meghatározott helyre publikálja a build-et.
- A Team Foundation Server data tier (adatbázis szerver) tölti be a data repository (adattárház) szerepét: itt tárolódnak a build részletei és a log event-ek (naplózott események), láthatóvá és elemezhetővé téve a buildet a Build kliens számára.

A következő ábra ezen komponensek egymással való együttműködését szemlélteti:



Megyjegyzés: A diagram csak egy logikai arhitektúra. A Team Foundation rendszer fizikailag máshogyan is strukturálható. Például: nagyobb projectek esetén logikus döntés egy dedikált Build Server létrehozása, de a Build Server ugyanazon a gépen is futtatható, ahol a TFS Application Tier, vagy esetleg még a Data Tier is. Nagyobb lélegzetvételű project-ek esetén pedig az a leghatékonyabb, ha egy egész számítógép farmunk van Build gépekből, és ezen futnak a Build server komponensei.

3.2.1. A Team Build kliens

A Team Foundation Build rendszer elsőszámú kliense a Team Explorer, mely a Visual Studio beépülő moduljaként, vagy önálló alkalmazásként telepíthető, használható. Az aktuálisan futó összes Team Project-hez tartozó Build elérhető, vezérelhető, konfigurálható a Build node-ból, mely a Team Explorerben található. Ezek az egységek úgy tekinthetőek, mint beállítások egy-egy halmaza, beleértve azt is, hogy mely file-k tartoznak a build-hez, melyek azok a test-ek, melyeket a build részeként futtatni kell , vagy hogy hova lesz a build publikálva, stb.

3.2.2. Az Application Tier

A Team Foundation Build 4 web serviceből áll, melyek az Application Tier Szerver-en futnak:

- Build Controller web service: Az API-t (Application Programming Interface) kiegészítve irányítja, kontrollálja a build-eket. Felelős többek közt azért is, hogy a build-et elindítsa és leállítsa, vagy jelezze, hogy véget ért a build.
- Build Store web service: Ennek a web service-nek köszönhető, hogy a build-del kapcsolatos adatokat a Data Tier-en el lehet menteni és újra elő lehet venni szükség esetén a későbbiek során.
- Build Integration web service: Lehetővé teszi, hogy a Build Server és más kliensek kapcsolatba kerüljenek más Team Project egyedekkel is.
- Publish Test Results web service: Ez az az API, mely megosztja a build test eredményeit a Project Team tagjaival.

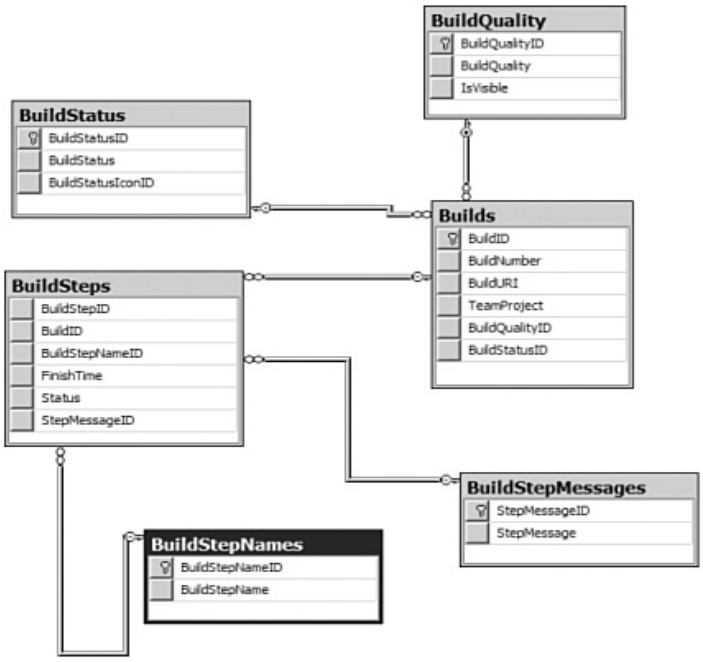
3.2.3. Build Server

A Build Server implementálja magát a Build processt. Windows service-kből áll (Team Build Service-k), amelyek az AT felől érkező utasításokat hajtják végre. Ennek a service-nek a feladata, hogy a Build Database-ből nyert információk szerint használja a mögöttes bulild motort (MSBuild).

3.2.4. Build Repository

A Build repository-t a TFS DT implementálja a saját SQL Server adatbázisa alapján, amit TFSBuild-nek hívunk. Ez az adatbázis több, mint 20 táblát tartalmaz, amik a build definíciókal, test eredményekkel, és work item-ekkel kapcsolatos információkat tartalmazzák.

A következő ábra az adat modelljét mutatja néhány fontosabb build-et definiáló táblának.



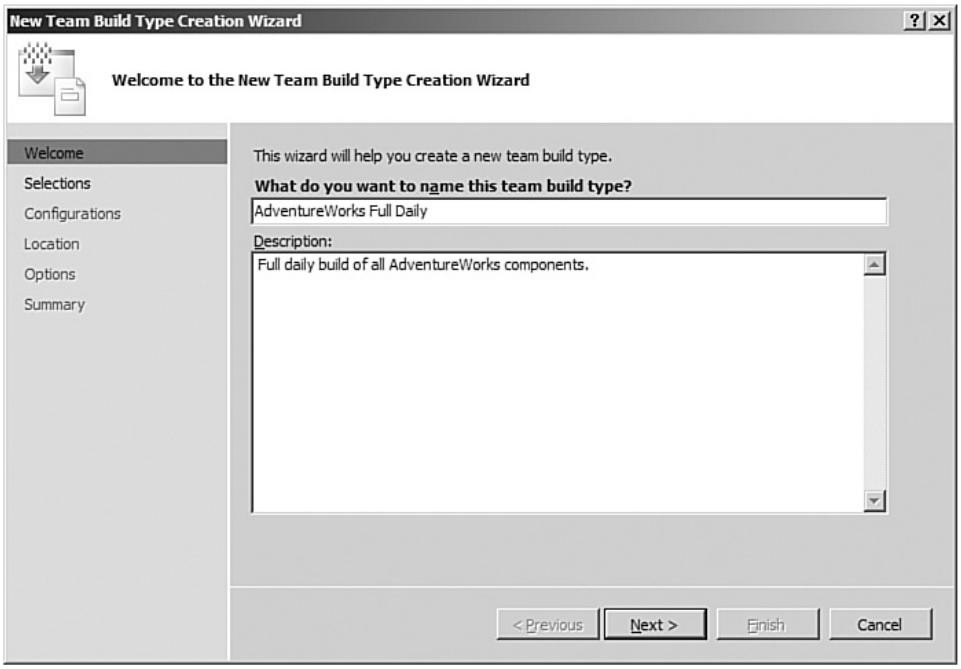
Most, hogy betekintést nyertünk a Team Build világába is, ideje, szeretném egy konkrét példával szemléltetni, hogyan lehet létrehozni, specifikálni, elindítani, majd pedig analizálni egy build processzt a Visual Studio Team System segítségével.

3.3. Új Team Build létrehozása

A Team Foundation Build bevezeti a build types (build típusok) fogalmát. Egy build type az adott build-del kapcsolatos összes konfigurációs adatot tartalmazza. Lényegében egy build összes alkotóeleme itt definiálódik. Egy build type-ot Team Explorer-en keresztül, a New Build Type Creation Wizard segítségével lehet létre hozhatun létre.

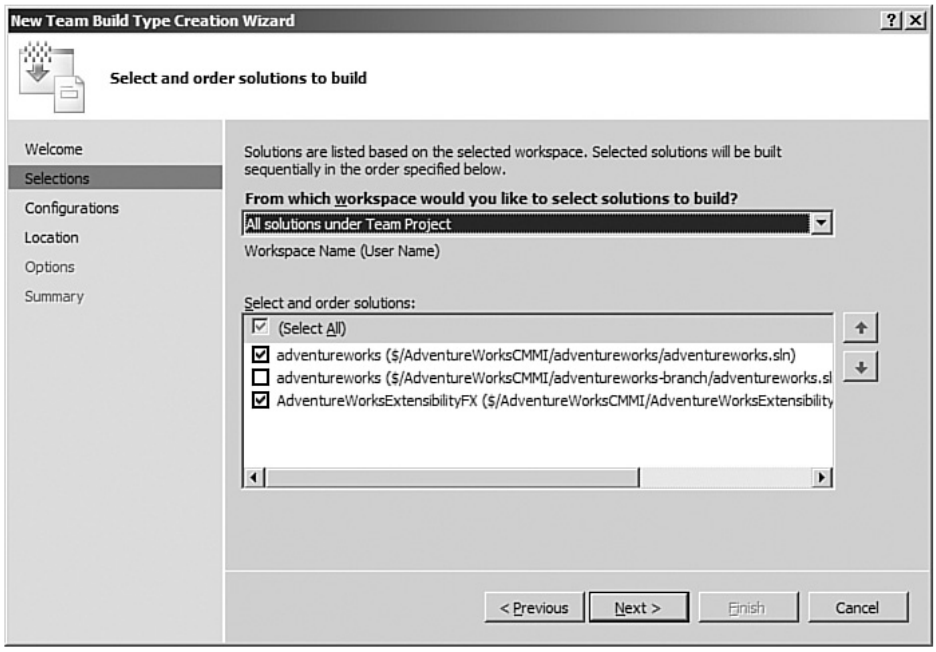
3.4. Build adatok specifikációja

Az első lépés, hogy megadjuk a build nevét. Ez ugyanaz a név, ami a Team Explorer Builds node-on belül és a build reports-ban is található.
A következő ábra a Build Type wizard első oldalát szemlélteti:

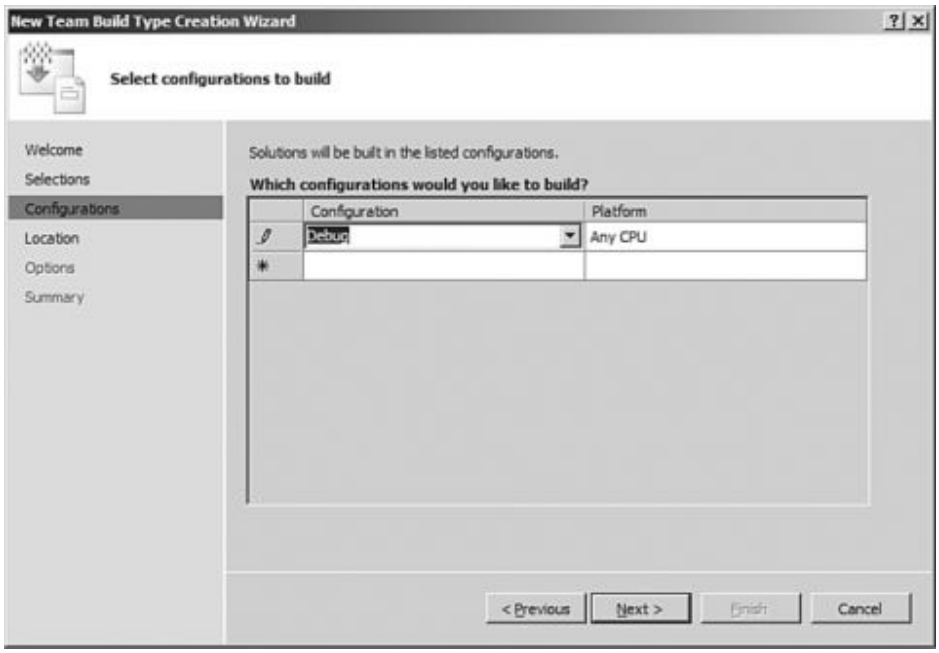


A varázsló következő lépése során választhatjuk ki azokat a project-eket, amelyeket build-elni akarunk. Erre való a legördülő lista az ablak tetején. Itt ki lehet választani az adott Workspace összes elérhető project-jét. Itt lehet meghatározni a build sorrendet is. Ez olyan esetben fontos, ha sorrendi függés van a különböző

solution-ok lefordítása által létrejövő binary-k között.

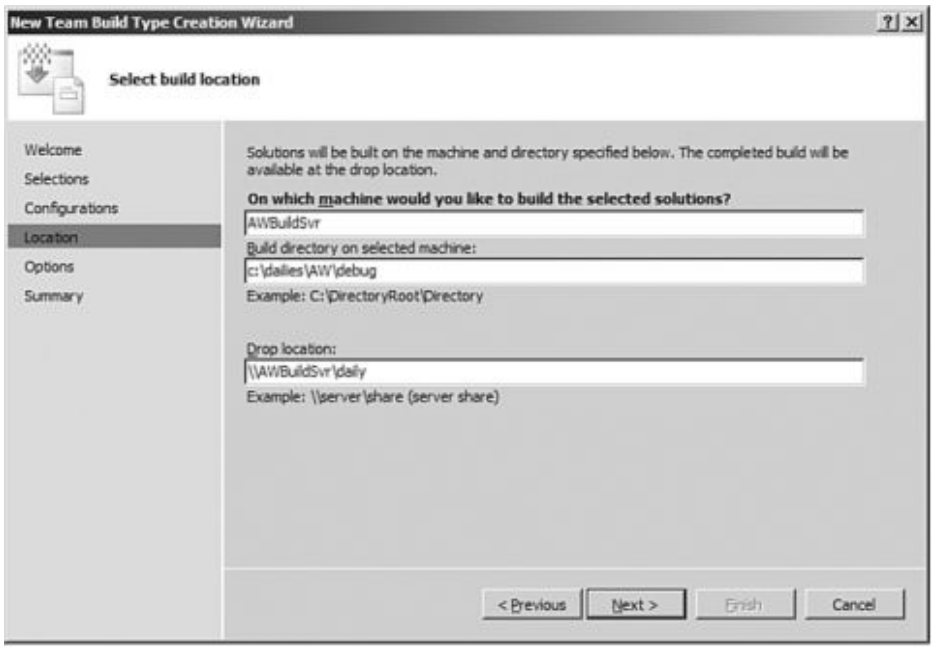


A Build konfiguráció adja meg a build paraméterét. Ez alapján dől el, hogy a build release vagy debug típusú-e, és szintén itt adható meg a build cél CPU platformja is.

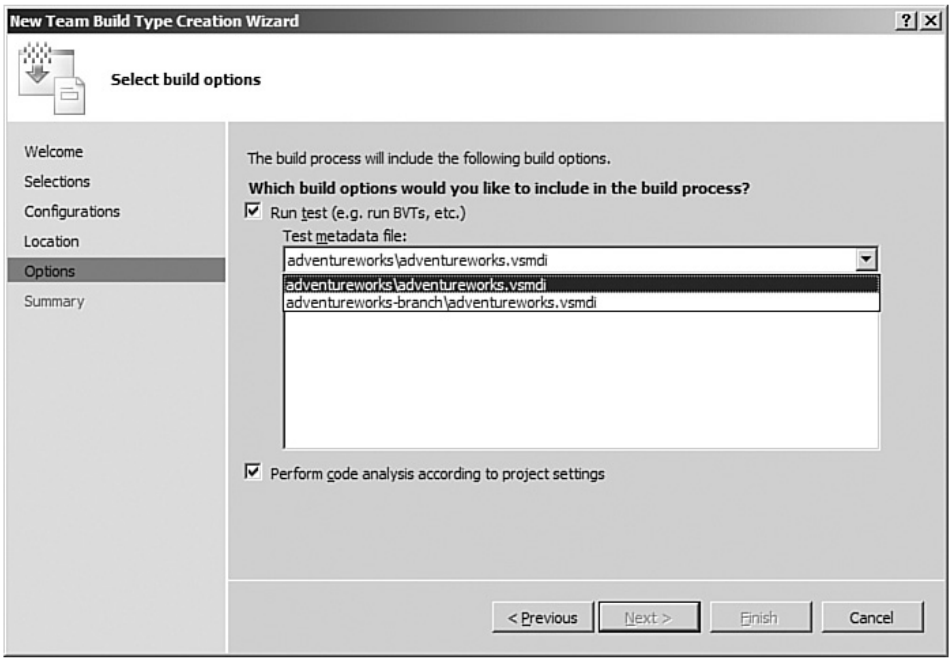


A TFB-dnek tudnia kell, hogy melyik szerver funkcionál Build Server-ként. Fontos,

hogy ekkorra már ez a szerver ki legyen jelölve Build Server-nek, ahogy azt a TFS install médián levő setup file futtatásakor tettük. Ennek során települt a Build Service, és a szolgáltatásnak futnia kell már a build elindítása előtt:



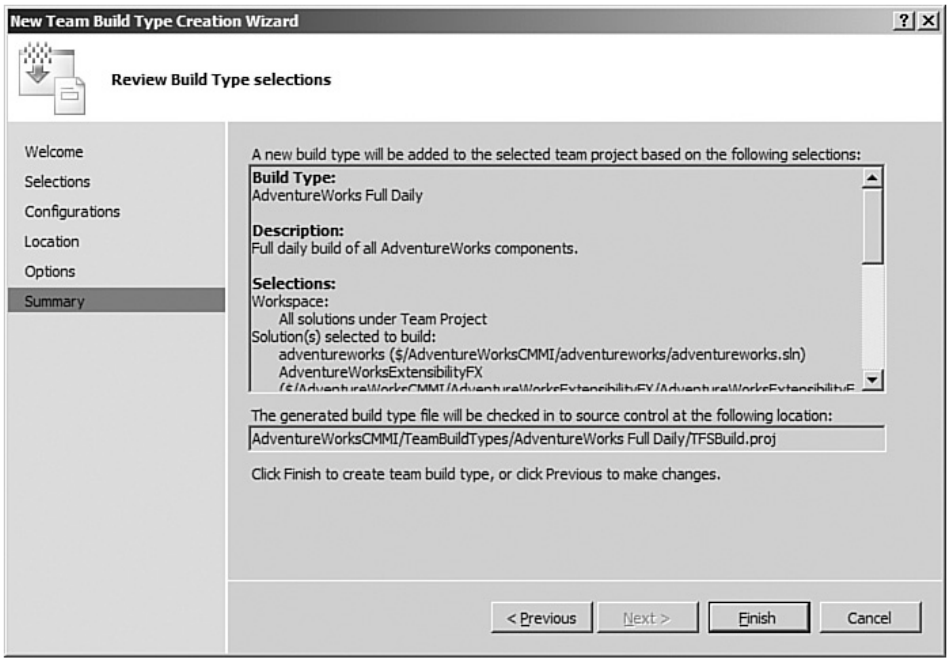
A build során teszteket is futtathatunk. A varázsló következő oldalán lehet megadni, hogy mely tesztek fussanak le a build alatt, vagy után. Az alsó checkbox használatával adhatjuk meg, hogy egy statikus kód analízis is lefusson a forrásfájlokon, amelyeket a build-ben megadtunk.



A VSTS 2008 esetében az új Build létrehozása némiképpen módosult a 2005-ös verzióhoz képest[5]. Lehetőség van korlátozni az adott időpillanatban szerveren tárolt Build-ek számát, melynek köszönhetően a régi build-ek kitakarításával jelentős tárhelyet tudunk felszabadítani. A szabály neve Build Retention Policy, és meg lehet adni a build végeredménye szeirnt is: failed (hibás), stopped (leállított), partially succeedded (részben sikeres), vagy sikeres build-ekre vonatkozóan. Ami viszont véleményem szerint még jelentősebb újítás: az ún. időzített Build-ek (Scheduled Build) és Build Triggerek létrehozása. Ez a VS 2008 Team System-en a varázsló utolsó lépése az összegzés előtt, és itt lehet beállítani, hogy mikor fusson le automatikusan a Build. Beállítható, hogy minden alkalommal build történjen, ha valaki forrásfile-t checkin-el a szerverre. Ennek a neve: Continous Integration. Szintén ennél a lépésnél állíthatjuk be azt is, hogy új build-ek a már futó build-ektől függően induljanak. Ebben az esetben, ha már fut build, akkor egy vagy több check-in esetén az új build nem indul el amíg a régi be nem fejeződött.

Ha a project szempontjából azt véljük hatékonyabbnak, akkor lehetőség van megadni, hogy a hét mely napjain és hány órakor induljon build. Tehát ha pl. nagyobb project esetén azt szeretnénk, hogy ne a munkaidőben használódjanak szerverünk erőforrásai a build processre, akkor úgy értdemes időzítenünk a build-et, amikor a legkevesebben használják a szervert.

A 6. és egyben utolsó lépés során a New Team Build Type Creation Wizard összegzi, hogy miket adtunk meg. Teszt futtatásához fontos, hogy fel legyen telepítve a Build Agent-re a Visual Studio Test Edition, a Code Analysis futtatásához pedig a Development edition jelenléte szükséges.



A Finish gomb megnyomása után két dolog történik:

- Létrejön egy XML file (TFSBuild.proj), amiben tárolódnak a build beállításai
- és hozzáadódik a Source Control System-hez, mint egy új project, aminek a neve megegyezik a Build nevével.
- Ha a későbbiek során szükséges a build type-ot szerkeszteni, módosítani, akkor a TFSBuild.proj file szerkesztésével lehet ezt megtenni. Erre remek eszköz a Visual Studio XML szerkesztője.

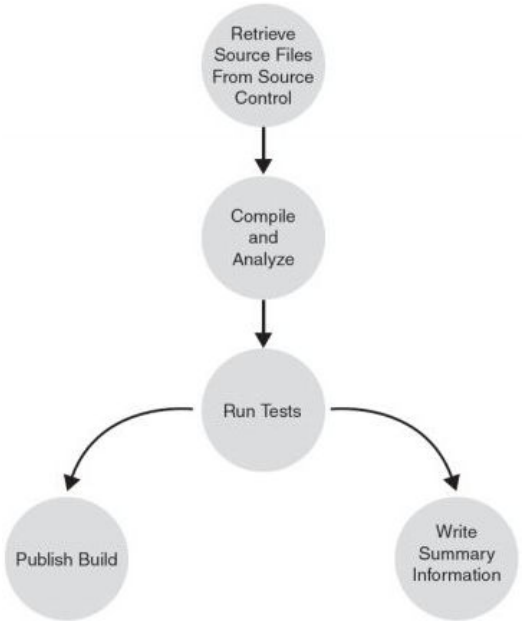
3.5. Build indítása

Ha van egy vagy több már definiált build, akkor a csapat tagjai manuálisan a Team Explorerből, vagy időzítve elindíthatják azokat amikor csak akarják. Eredetileg a TFB nem képes rá, hogy időzített build-eket hajtson végre egyszeri vagy többszörösen ismétlődő rendszerességgel. A TFS parancs segítségével azonban parancssorból használhatjuk a standard window scheduler task-ot. Egy időzített build beállításához a következő információkra van szükség:

- a TFS AT server URL-je
- a build type-ot tartalmazó Team Project neve
- a build type neve, amit schedule-álni szeretnénk

A parancs neve, amivel végrehajtani és irányítani lehet a build-et: TFSBuild.exe (TeamBuildInstallDir\Comon7\IDE\). Néhány paraméter megadásával ez a .exe kapcsolódik a build serverhez, lokalizálja a team project-et, letölti a konfigurációval kapcsolatos információkat, majd pedig végrehajtja azt.

A következő ábra egy hagyományos build processz futását szemlélteti:



3.6. Buildek elemzése és nyomonkövetése

A build alatt az engine folyamatosan naplózza az build során zajló történések adatait. A logok visszakerülnek a Build Repository-ba, és így a csapattagok számára lehetővé válik, hogy nyomonkövssék a Visual Studio-val az események alakulását. A Build információk a Team Bulid Exploreren keresztül láthatóak. Ez egy böngészőablak, amely listázza a befejezett, vagy épp zajló Build-eket. A Team Build Explorer az elsőszámú eszköz a build folyamatának, vagy a már teljesített összegzések szemrevételezésére.

Minden Build jelentés a VS document ablakában nyitható meg. A riport következő komponensekből tevődik össze:

- Summary (Összegzés): összegzi a build részleteit, a build-el kapcsolatos adatokat, mint pl a build nevét, a személyt, aki indította, a gépet, ami futtatta, a jelenlegi állapotát, és a linket a build log-hoz.
- Build Steps (Build lépések): tartalmazza a listát (ha a build még zajlik, akkor dinamikusan), ami megadja a dátumot és a pontos idejét a build minden fázisának.
- Result Details (Eredmény részletek): a build által generált hibákat és figyelmeztetéseket tartalmazza, valamint a build-del futó tesztek eredményeit.
- Associated Changesets (hozzárendelt változtatások): a build-ben részt vevő összes changeset listáját tartalmazza hyperlink-el együtt.
- Associated Work Items (hozzárenelt munkadarabok): ez az összes build-hez rendelt workitem-et sorolja fel hyperhivatkozásokkal.

A Build logok és Reportok segítségével theát a csapat rendelkezésére áll minden infromáció, amely egy esetleges sikertelen build-et okozhat, és sikeres build esetén is fontos adatokat tudhatunk meg az alkalmazással, az alkalmazás teljesítményével, állapotával kapcsolatban.

3.7 Gyakorlati példa

Annak érdekében, hogy az elhangzott elméleti jellegű információkat könnyebben elképzelhesse a Tisztelt Olvasó, szeretnék picit gyakorlatiasabb dolgokról is írni, amelyet az elhangzottak összegzésének szánok. Ennek érdekében most egy Team Project létrehozásának módját szeretném lépésről lépésre ismertetni egészen a build folyamat befejeződéséig.

Tételezzük fel, hogy a csapat számára egy .Net alkalmazás elkészítése a feladat. Annak érdekében, hogy a leghatékonyabban, tehát a csapat valamennyi tagjának bevonásával el tudjunk kezdeni dolgozni a feladaton, szeretnénk egy Team Project keretein belül szervezni a teendőket. Miutrán Visual Studio-val létrehoztuk a .Net Solution-t, nincs más teendőnk, mint hogy megírjuk a public property-ket és metódusokat (ezt a Class Designer segítségével egyszerűen megtehetjük), mielőtt a Visual Studio Team System-en belül a solution-hoz létrehozunk, és hozzáadunk egy Unit Test Project-et. Erre a Team Project-ünk buildelésénél lesz szükségünk, hiszen egység teszteket szeretnénk majd futtatni a build során.

Ha már van egy működőképes Team Foundation Server telepítésünk, ami a hálózatra van kötve, a szerverhez való kapcsolódást a Visual Studio Tools menüjének Connect To Team Foundation Server parancsával oldhatjuk meg. Kapcsolódás után már létre tudunk hozni egy új Team Projectet. Általában a Team Project-et még kód írás előtt értemes létre hozni. A project létrehozásának első fázisában ki kell választanunk a Process Template-et. Ez fontos lépés, hiszen a template határozza meg a workitemeket, report-okat és a biztonsági beállításait a project-nek.

A következő lépésben beállítjuk a Source Control-t a project-hez. Érdemes egy üres source control mappát létrehozni s elnevezni a project nevével megegyezően. Ezután összegezve látjuk majd megadott beállításainkat. A project létrehozása következik, ami eltart egy darabig, mivel sok dolgot kell a TFS-nek tennie. Többek között ekkor jön létre a Project-ünk WSS portálja is. Ezekután megadhatjuk a biztonsági beállításokat, amivel definiáljuk a csapat tagjainak szerepköreit is. Tehát már létrehoztunk egy .Net solution-t, hozzáadtuk a Unit Testeinket és létrehoztunk egy új Team Project-et kijelölve a fejlesztő, tesztelő és egyéb szerepköreit, beállítva a

jogosultságokat.

A következő lépés során elkészítjük a Unit Test-eket úgy, hogy távoli gépről is futni tudjanak a Team Foundation Build Server-en. A Visual Studio Team System-en belül, a Test Manager automatikusan létrehozza a teszt metadata file-t, .vsmdi kiterjesztéssel. Ez tartalmazza a unit test konfigurációt a project számára, ami biztosítja, hogy a teszt fusson távoli build során is. Ezek az adatok a .vsmdi file-ban mentődnek el, és a file szintén beadásra kerül a Source Control System-be.

Készen állunk a project build type-jainak definiálására. Ezt a Team Explorer-ből a Team Build opciónál érhetjük el, a Create New Build types paranccsal. Egy olyan build-et hozzunk létre, ami lefordítja a .Net Solution-unkat, és a Unit Test Project-ünket is. A már ismertetett varázslóban meg kell adnunk a build-elésre szánt project-ek neveit, a Build Server-t, és be kell állítanunk, hogy a Unit Test-ek fussanak is le a build során. Miután a build befejeződött, rendelkezésünkre állnak a Build Report-ok, amiből kiderül, hogy ki indította az esetlegesen sikertelen Build-et. A build naplózott adatairól email-t is külhet a rendszer a fejlesztőkne, benne az adatokkal, melyekkel fény derülhet a hibás build, vagy teszteset okára. Az AssignWorkItem funkcióval egy feladatot (Task) adhatunk a fejlesztőnek, aki checkin-elte a problémát okozó kódot.

Nagy vonalakban ezek a visszatérő mozzanatai egy tipikus Visual Studio Team Systemmel fejlesztett Team Project életciklusának, de megjegyezném, hogy magát a fejlesztést, ami az egész Project lényege, most szándékosan nem részleteztem, mert az nem ennek a dolgozatnak a témája.

4. Befejezés

Dolgozatom céljául a Visual Studio Team System és szerveralkalmazásának, a Team Foundation Servernek átfogó leírását, a kapcsolódó szerepkörök bemutatását tűztem ki. Írásom alapján a Tisztelt Olvasónak lehetősége nyílt megismerni a VS Team System elődjét, különböző kiadásait, továbbá mélyebben érintettem a szoftverfejlesztés lényegi célját, a Build folyamatot is. Összegzésként végigkövettük egy tetszőleges alkalmazás életciklusának fázisait, ezáltal gyakorlati szemszögből még jobban megvilágítva ennek a meglehetősen összetett rendszernek egy ajánlott használati módját, üzemeltetését.

Tisztában vagyok vele, hogy az érintett területek tárgyalása során sok esetben hagytam nyitott kérdéseket magam mögött. Számos olyan terület is lenne még, amelyet meg sem említettem, pedig talán szorosan kapcsolódna a témához. Amennyiben a Kedves Olvasó így érez, azt kell mondjam, hogy dolgozatom elérte célját. Naívság lett volna ugyanis részemről egy hiánytalanul teljes kép visszaadásával próbálkozni egy ily mértékben testreszabható, ráadásul bővíthető rendszerről a rendelkezésemre álló terjedelemkorlátokon belül. Dolgozatom elolvasása után remélem, hogy lesznek, akik lázas keresésbe kezdenek egy-egy általam érintett témát még jobban körüljárni. Hiszen az informatika fejlődése, s így a fejlesztői környezetek fejlődése is megállíthatatlan. A Visual Studio 2010 és a .Net Framework 4.0 már elérhető tesztelés céljából, és a többi mammutcég „gyártósoráról" is sorozatban potyognak le az újabbnál újabb fejlesztői eszközök. A verseny tehát e téren megállíthatatlan, s igazi győztesének csak magukat a fejlesztőket mondhatjuk, akiknek egyre hatékonyabb, produktívabb alkalmazások állnak rendelkezésükre, hogy a mai modern szoftverpiac magas elvárásait teljesíteni tudják.

5. Köszönetnyilvánítás:

Köszönetet mondok Dr. Kuki Attila témavezetőmnek, hogy lehetőséget biztosított munkám sikeres elvégzéséhez és dolgozatom megírásához. Köszönöm segítőkész támogatását és dolgozatom alapos és kritikus átnézését.

Köszönetet mondok továbbá Jagó Jácint külső konzulensemnek, hogy az IT Services Hungary Kft. Microsoft Fejlesztői csapata infrastrukturális hátterének megismerésében segítségemre volt, s ezáltal segítette munkám elvégzését. Köszönet a konzultációval töltött időért, és az építő kritikákért, melyekkel segített dolgozatom színvonalát tovább emelni.

6. Források

[1] Richard Hundhausen – Working with Microsoft Visual Studio 2005 Team System

[2] http://msportal.hu/blogs/smulo/archive/2007/10/17/team-foundation-server-2008-mi-233-rt-is.aspx

[3] http://msdn.microsoft.com/en-us/library/ms181369(VS.80).aspx

[4] Lars Powers, Mike Snell - Microsoft® Visual Studio 2005 Unleashed

[5] Lars Powers, Mike Snell - Microsoft® Visual Studio 2008 Unleashed

[6] Official Microsoft Visual Studio Documentation

[7] http://msdn.microsoft.com/en-us/library/cc304371.aspx