Csutorás Zoltán, Árvai Zoltán, Novák István

A Scrum keretrendszer és agilis módszerek használata a Visual Studióval

A szerzők a könyv írása során törekedtek arra, hogy a leírt tartalom a lehető legpontosabb és naprakész legyen. Ennek ellenére előfordulhatnak hibák, vagy bizonyos információk elavulttá válhattak.

A példákat és a módszereket mindenki csak saját felelősségére alkalmazhatja. Javasoljuk, hogy felhasználás előtt próbálja ki és döntse el saját maga, hogy megfelel-e a céljainak. A könyvben foglalt információk felhasználásából fakadó esetleges károkért sem a szerző, sem a kiadó nem vonható felelősségre.

A könyv a devportal.hu webhelyről ingyenesen letölthető.

Tartalomjegyzék

1. Az agilis módszerek

Ebben a fejezetben az alábbi témákat ismerheted meg:

- A vízesés modell kihívásai.
- Az agilis termékfejlesztés alapelvei.
- A legelterjedtebb agilis módszerek.

Fontos leszögezni, hogy az agilis szoftverfejlesztés nem módszertan, hanem egyfajta hozzáállás a szoftverfejlesztési projektek során jelentkező feladatok megoldásához, a megrendelő és a fejlesztők közötti együttműködéshez és a fejlesztő csapatok tagjainak közös munkájához.

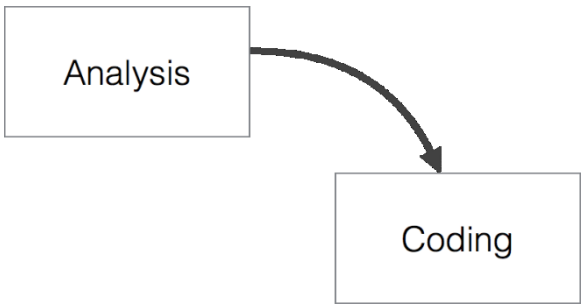
Az elmúlt néhány évben az agilis fejlesztés népszerűsége töretlenül növekszik, egyre több fejlesztői csapat tűzte ki az „Agilisan fejlesztünk!" jelszót a zászlajára. Sajnos ezek közül a csapatok közül csak kevés olyan van, amelynek tagjai pontosan értik az agilis szemlélet lényegét. Ebben a fejezetben az agilis fejlesztés legfontosabb alapelveinek és irányzatainak bemutatásával igyekszünk tisztázni az agilis szoftverfejlesztés körüli zavaros képet.

A tradicionális modellek kihívásai

A vízesés modell az elmúlt két évtized domináns szoftverfejlesztési irányzata volt, és a mai napig is népes követőtábora van. Miért foglalkozunk mégis az agilis módszerekkel? Mi a baj a vízesés megközelítéssel?

A vízesés módszer

Dr. Winston W. Royce írta le a ma ismert vízesés modell alapelveit a „*Managing the development of large software systems*" (Nagy szoftverrendszerek fejlesztésének menedzsmentje) c. publikációjában. Ekkor 1970-et írtunk. Önmagában az a tény, hogy több mint 40 éves koncepcióról beszélünk, még nem jelenti azt, hogy probléma van az elképzeléssel. Royce alapgondolata az volt, hogy bármekkora szoftverfejlesztési projektet két élesen elkülönülő szakaszra kell bontani: az elemzésre és az implementációra (1-1 ábra).

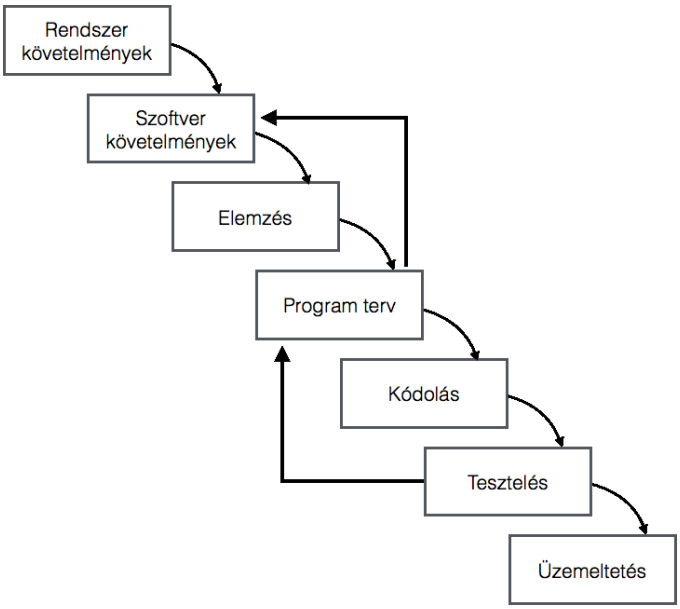


1-1 ábra: Elemzés, implementáció

Ezt az egyszerű alapelvet bontotta további alapvető lépésekre, melyek során a szoftver az elképzelésből egy üzemeltethető, működő rendszerré alakul (1-2 ábra).

A modell alapja, hogy a rendszerrel szemben támasztott követelményeket képesek vagyunk a projekt elején felmérni és ennek alapján megtervezni egy olyan szoftvert, mely alkalmas ezen követelmények kielégítésére. A megközelítés logikus, jól követi a hagyományos mérnöki gondolkodást.

Miért foglalkozunk ennyit a vízesés modellel egy agilis szoftverfejlesztésről szóló könyvben? Azért, mert maga Dr. Winston W. Royce is így fogalmaz a módszer bemutatása során:



1-2 ábra: A vízesés modell

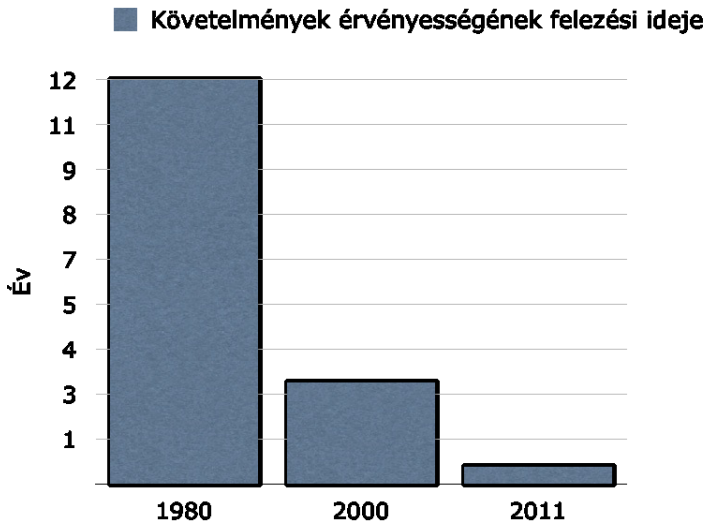
> *„Én hiszek ebben a módszerben, de a fent bemutatott implementáció kockázatos és potenciálisan hibákat hordoz magában. A problémát az jelenti, hogy az első alkalom, amikor a megoldást ellenőrizhetjük a tesztelési fázisban van, ami viszont a folyamat legvégén áll. (…) Ennek eredményeként a fejlesztési folyamatot újra kell kezdeni, ami akár 100%-os költség/határidő túllépést is eredményezhet."*

Éppen ezért nevezte a fenti modellt egyszerűsített modellnek, melyet ő maga is alkalmatlannak ítélt komplexebb szoftverek fejlesztésére, s egy visszacsatoláson és prototípus készítésen alapuló tervezési fázissal javasolta kiegészíteni. Sajnálatos módon ez a továbbfejlesztett modell nem szerzett akkora ismertséget, mint az egyszerűsített modellje.

Az elmúlt néhány évtized szoftverfejlesztéseinek eredményeit több kutatás is vizsgálta. Ezek mindegyike egyetért abban, hogy a szoftverfejlesztési projektek rendszeresen átlépik a tervezett költség- és határidőkeretet, miközben a projekt elején elképzelt szkóptól jelentős mértékben eltérnek.

Követelmények instabilitása

Mi a gond a vízesés modellel? Ahogyan Dr. Winston W. Royce is megsejtette, a probléma elsődleges forrása a követelmények megismerhetőségében és állandóságában rejlik. Egy 2012-es kutatás szerint (1-3 ábra) az üzleti követelmények felezési ideje radikálisan csökkent az elmúlt 30 évben.



1-3 ábra: A követelmények felezési ideje

A kutatás azt vizsgálta, hogy a ma érvényes üzleti követelmények fele mennyi idő alatt válik érvénytelenné, azaz a – radioaktív anyagok radioaktivitásának csökkenésének jellemzéséhez alkalmazott felezési időhöz hasonlóan – mennyi az üzleti elképzelések érvényességének felezési ideje. Ezek alapján:

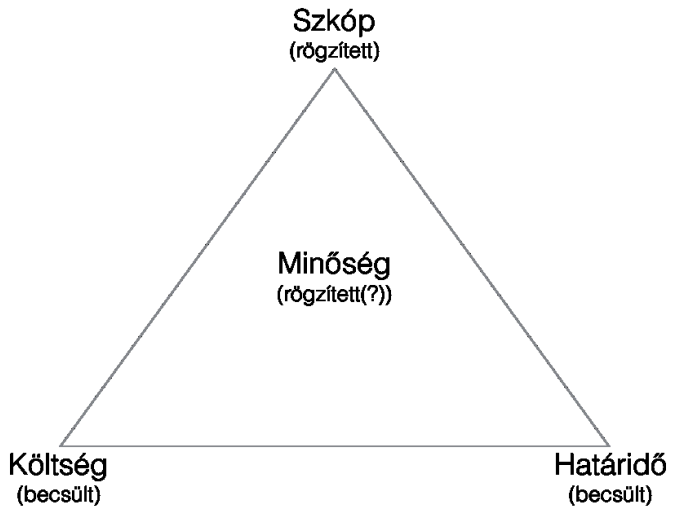
- 1980-ban 12 év alatt vált érvénytelenné a megfogalmazott üzleti igények fele.
- 2000-ben a követelmények felezési ideje már csak átlagosan kb. 3 év volt.
- 2011-ben pedig egy átlagos üzleti környezetben a célkitűzések és követelmények fele 6 hónap alatt elavult.

Forrás: University of Missouri (Kansas City)

Mit jelent ez egy szoftverfejlesztési projekt esetében? Azt jelenti, hogy ha egy fejlesztési projektet a klasszikus vízesés modell szerint szeretnénk megvalósítani, akkor kevesebb mint 6 hónapunk van arra, hogy elkészítsük a szoftverünket részletes felméréssel, teszteléssel, üzemeltetésre átadással együtt. És még ekkor is a funkcionalitás fele érvénytelenné válhat a megváltozott felhasználói igények következtében.

Műszaki adósság

Valószínűleg sokunknak nem ismeretlen a projektmenedzsment vasháromszögének is nevezett ábra (1-4 ábra), mely a szoftverfejlesztési projekteket három fő kritérium és a minőség mentén határozza meg.



1-4 ábra: A projektmenedzsment vasháromszöge

A klasszikus vízesés módszerekben tehát feltételezzük, hogy ismerjük a rendszerrel szemben támasztott követelményeket (értsük most ezt a projekt szkópja alatt), és a szakértelmünk alapján megbecsüljük a projekt várható idő- és ráfordítás (költség)-szükségletét. De mi történik, ha valamit mégis elrontottunk? Mi van, ha túl optimisták voltunk? Mit tegyünk, ha a megrendelő oldal és a fejlesztő oldal mást értett egy követelmény teljesülése alatt? A három sarokpont közül általában legalább kettő (a költség és a határidő) mozogni fog. Méghozzá kedvezőtlen irányba.

Ha ez mégsem lehetséges, akkor a fejlesztő csapat egyetlen paraméteren tud még spórolni, ez pedig a minőség. Rövidtávon a szoftver minőségének feláldozásával viszonylag jelentős költség és határidő nyereségre lehet szert tenni. Rövidtávon! Ezt nevezzük műszaki adósságnak.

A műszaki adósság jelei:

- gyakori hibák,
- „spagetti kód”,
- automatizált tesztek hiánya,
- dokumentációk hiánya.
- fejlesztők „saját kódjai” (csak XY érti)
- és végül: egyre lassuló fejlesztés, elégedetlenség, magas fluktuáció.

Mindez kezdetben csak ártalmatlannak tűnő „release előtti hajtásnak” indul, később azonban ellehetetleníti a teljes alkalmazásfejlesztést. A legtöbb szoftverfejlesztési projekt hatalmas műszaki adóssággal küzd.

Az agilis fejlesztés alapelvei

A ’90-es évek végére az objektum orientált szoftverfejlesztés és a hatékony fejlesztői eszközök újra felvetették a lehetőségét annak, hogy a követelmények teljes körű, előzetes feltárása helyett (ami nem működött) esetleg hatékonyabb lenne az implementációs fázist előbbre hozni és a követelmények feltárásának részévé tenni a gyakori prototípuskészítéssel és a felhasználóknak való bemutatással. Ez sokak számára a mai napig eretnek gondolatnak számít, hiszen éppen az ilyen jellegű hozzáállás miatt alakult ki a ’70-es évek szoftverkrízise, amikor tömegesen állították le hatalmas veszteségekkel a nagyobb fejlesztési projekteket. És ezt a krízist orvosolta valamennyire a vízesés modell elterjedése.

Mégis rengetegen kísérleteztek *iteratív* (vagy – ahogy sokan nevezik – *adaptív*) módszerekkel, melyekkel egészen szép sikereket értek el. Ilyenek voltak a *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD), *Dynamic Software Development Method* (DSDM) vagy az *Extreme Programming* (XP), a *SCRUM* vagy a *Lean/Kanban* keretrendszerek.

Az agilis kiáltvány

Végül, 2001-ben, a fenti agilis irányzatok néhány jeles képviselője megfogalmazta az agilis kiáltványt (*Agile Manifesto*), mely azokat az alapelveket tartalmazta, melyekben mindannyian hisznek és egyetértenek, és melyek meggyőződésük szerint sikeresebb szoftverfejlesztési projekteket eredményezhetnek.

> *Az agilis kiáltvány*
>
> *A szoftverfejlesztés hatékonyabb módját tárjuk fel saját tevékenységünk és a másoknak nyújtott segítség útján. E munka eredményeképpen megtanultuk értékelni:*
>
> ***Az egyéneket és a személyes kommunikációt a módszertanokkal és eszközökkel szemben***
> ***A működő szoftvert az átfogó dokumentációval szemben***
> ***A megrendelővel történő együttműködést a szerződéses tárgyalással szemben***
> ***A változás iránti készséget a tervek szolgai követésével szemben***
>
> *Azaz, annak ellenére, hogy a jobb oldalon szereplő tételek is értékkel bírnak, mi többre tartjuk a bal oldalon feltüntetetteket.*

Mit takarnak ezek a mondatok? Az elmúlt évtizedben a szoftverfejlesztői szakma egyik leginkább félreértelmezett eleme az agilis kiáltvány. Ahhoz, hogy megértsük, végig kell gondolnunk azt a közeget, amiben ez a néhány mondat született.

Mivel a vízesés modell szigorú módszertanisága mellett sem sikerült orvosolni a fejlesztési projektek legtöbb problémáját, ezért a vezetők egyre szigorúbb, egyre adminisztratívabb intézkedéseket kezdtek szorgalmazni. Ilyenek voltak a még részletesebb projekt- és rendszertervek, még több dokumentáció, még szigorúbb szerződések. Ezek viszont további erőforrásokat vontak el a produktív fejlesztői csapatoktól, amitől a fejlesztés csak egyre drágább lett, a határidők egyre szűkösebbek, és a szerződések egyre nehezebben váltak tarthatóvá.

Végül senki nem tartotta be a szigorú módszertani előírásokat, így nem sikerült kiaknázni annak még néhány előnyét sem. A fejlesztési iparág újra az anarchia felé kezdett sodródni. Aki megpróbálta tartani magát a szigorú adminisztratív eljárásokhoz, az rendkívül drágán tudott csak dolgozni, és egyetlen ponton tudott spórolni. Műszaki adósságot halmozott fel. Az ördögi kör bezárult…

Ebben a környezetben született meg az agilis kiáltvány olyan szakemberek tollából, akik megelégelték ezt a helyzetet. A kiáltvány célja, hogy az eszközök helyett koncentráljunk újra azokra az alapértékekre, amelyek támogatására létrehoztuk őket. Tehát:

- Az agilis kiáltványt olyanok írták, akik a gyakorlatban, valós projekteken dolgoznak. Az agilis szemlélet alapja tehát a gyakorlati tapasztalat.
- A kiáltvány jobb oldalán szereplő elemeknek is megvan a maguk értéke, de nem szabad elfelejteni azt, hogy a bal oldali elemeket kellene szolgálniuk.
- A szoftverfejlesztés lényege, hogy megértsük a megrendelői igényeket és kreatív megoldásokat dolgozzunk ki ezek kielégítésére. Ennek elsődleges módja a jó csapatmunka és tehetséges, motivált egyének együttműködése. A módszertanok egyébként pont ezt az együttműködést igyekeznek segíteni azzal, hogy szerepköröket, munkaanyagokat és folyamatokat definiálnak. Lehet fontos a módszertan, de a jó együttműködés a fontosabb.
- A dokumentáció célja az, hogy a szoftver elkészítését, továbbfejlesztését, használatát és üzemeltetését lehetővé tegye. A dokumentáció tehát nem szabad, hogy öncélú legyen, a jó szoftver nem helyettesíthető semmilyen dokumentációval.
- A szerződéseket azért készítjük, hogy szabályozzuk két fél együttműködését. Ha a szerződés az együttműködés ellen hat, akkor nem teljesíti azt a célját, amiért elvileg létrehoztuk. A jó szerződés tehát az együttműködést, és nem a fegyverkezést szolgálja.
- A tervezés elsődleges célja a probléma feltárása és megoldási lehetőségek lefektetése. Ha a projekt során olyan információkat tárunk fel (és lássuk be, hogy általában ez a helyzet), ami az eredeti terveinkbe nem illeszkedik, akkor nem ezeknek az új információknak a semmibevétele a helyes megoldás, hanem a tervek módosítása. Nincsen semmi baj a tervekkel, azokra szükségünk van, de ne felejtsük el, hogy azok is a tanulást szolgálják! Ha újat tanultunk, akkor a terveknek ezt követniük kell.

Ha ilyen szemmel vizsgáljuk az agilis kiáltványt, akkor már rögtön nem valami csodabogárnak tekintjük, hanem logikus, értékes megközelítésnek.

A 12 agilis alapelv

Rendben van, hogy fontosak az alapértékek és a tradicionális módszertani eszközeink csak ezek szolgálatában állhatnak, de hogyan lesz ebből sikeres szoftverfejlesztés? Az agilis szoftverfejlesztés sokkal inkább szervezeti kultúra, mintsem egységes módszertan. Ennek a kultúrának az alapjait a 12 agilis alapelv fekteti le. Ezek együttes alkalmazása biztosítja azt, hogy a fejlesztő csapatok hatékony, sikeres fejlesztési projekteket valósíthassanak meg. Nézzük meg, melyek ezek!

> 1. Legfontosabbnak azt tartjuk, hogy az ügyfél elégedettségét a működő szoftver mielőbbi és folyamatos szállításával vívjuk ki.

Tehát az agilis szoftverfejlesztés első számú szabálya, hogy a csapat a lehető leghamarabb olyan működő szoftvert szállít, amit a megrendelő a lehető leggyorsabban a gyakorlatban kiértékelhet. Minden tevékenységnek ezt a célt kell szolgálnia.

> 2. Elfogadjuk, hogy a követelmények változhatnak akár a fejlesztés vége felé is. Az agilis eljárások a változásból versenyelőnyt kovácsolnak az ügyfél számára.

Az agilis fejlesztői csapatok tisztában vannak azzal, hogy a követelmények teljes körű felmérése nem lehetséges, ezért elfogadják azok változásának a lehetőségét. Ahelyett, hogy a kőbe vésett követelményekre alapozva terveznék és fejlesztenék a rendszert, felkészülnek azok változására. Ebből következik az agilis szoftverfejlesztés legfontosabb tervezési alapelve: a rendszer felépítése és tervezése során elsősorban a kód világos szervezésére és a lehető leggyorsabb változtathatóságára helyezik a hangsúlyt. Az agilis fejlesztés tehát adaptív (alkalmazkodó) megközelítést alkalmaz, szemben a vízesés modell prediktív (meghatározott) fejlesztési szemléletével.

> 3. Szállíts működő szoftvert gyakran, azaz néhány hetenként vagy havonként, lehetőség szerint a gyakoribb szállítást választva!

A harmadik alapelv lényege, hogy az agilis fejlesztés során rendszeres időközönként (kevesebb, mint 1 hónapos átfutással) működő terméket kell tudni leszállítani. Ez egy sor szoftverfejlesztési módszert (folyamatos integráció, automatizáltság stb.) megkövetel, melyekről a könyv későbbi fejezeteiben részletesen írunk.

> 4. Az üzleti szakértők és a szoftverfejlesztők dolgozzanak együtt minden nap, a projekt teljes időtartamában!

Ez az alapelv az agilis fejlesztői csapatok szervezésére, összeállítására gyakorol jelentős hatást. Az agilis fejlesztés csak úgy lehet működőképes, ha a fejlesztőkkel szorosan együttműködnek azok az emberek, akik a rendszerrel szemben támasztott felhasználói elvárásokat értik. A követelménykezelés tehát a fejlesztési folyamat integrált része, nem pedig egy azt megelőző, elkülönülő fázis.

> 5. Építsd a projektet motivált egyénekre! Biztosítsd számukra a szükséges környezetet és támogatást, és bízz meg bennük, hogy elvégzik a munkát!

A szoftverfejlesztés nehezen kontrollálható és irányítható tevékenység. Mivel minden egyes fejlesztő naponta többször találkozik új, megoldandó problémákkal, rendszeresen kell valamilyen implementációs döntést hoznia. A fejlesztés tehát nem tervezhető meg részletesen, éppen ezért nem szigorúan kontrollálható folyamat. A siker kulcsa egyedül az lehet, ha a csapat tagjai motiváltak és elszántak a felmerülő problémák megoldása iránt. Az agilis fejlesztési kultúrában a menedzsment feladata az ilyen csapatok létrehozása és a motivált munkavégzéshez szükséges feltételek megteremtése. A fejlesztői csapat bár nagy önállósággal bír, a felelőssége is nagy, és az előző alapelvből következő rendszeres leszállítás miatt nagyon jól mérhető és értékelhető a teljesítménye.

> 6. Az információ átadásának leghatásosabb és leghatékonyabb módszere a fejlesztői csapaton belül a személyes beszélgetés.

A fejlesztői csapat tagjainak szoros együttműködésben kell dolgozniuk, közösen kell problémákat megoldaniuk, és hatalmas mennyiségű információt kell megosztaniuk egymással. Ennek a leghatékonyabb módja az élőszóban történő, személyes beszélgetés. Ne tévesszen meg senkit ez alapelv! A személyes beszélgetések során vizsgált alternatívákat, döntéseket ugyanúgy dokumentálni kell annak érdekében, hogy a későbbi időszakban ezek visszakereshetőek, betarthatóak maradjanak! Ebből az alapelvből következik az az agilis csapatszervezési gyakorlat, hogy a fejlesztői csapatok tagjait lehetőleg közös területen, irodában helyezik el azért, hogy a személyes beszélgetésekre a lehető legkönnyebb legyen alkalmat találni.

> 7. Az előrehaladás elsődleges mércéje a működő szoftver.

Az agilis fejlesztői csapatok nem keresnek és nem is találhatnak mentséget arra, ha a munkájuk nem eredményezett működő szoftvert. Bármilyen jó is a csapatmorál, bármilyen ígéretes is az alkalmazott architektúra, az első számú mérce az, hogy a csapat képes-e rendszeres időközönként működő, integrált rendszert szállítani.

> 8. Az agilis eljárások a fenntartható fejlesztést pártolják. Fontos, hogy a szponzorok, a fejlesztők és a felhasználók folytonosan képesek legyenek tartani egy állandó ütemet.

Ez az alapelv is állandóságot, a rendszeres szállítási képességet helyezi az előtérbe. A fenntartható munkatempó egyrészt a csapatok tagjainak kiégését akadályozza meg – és egyben a motivált egyénekből álló csapat működőképesen tartásához is elengedhetetlen –, másrészt, a magas kódminőséget is szolgálja. A műszaki adósság felhalmozásának elsődleges oka a csapat siettetése, trehány munkába kényszerítése.

> 9. A műszaki kiválóság és a jó terv folyamatos szem előtt tartása fokozza az agilitást.

Ez a leggyakrabban elhanyagolt agilis alapelv. A gyakori kibocsátások, a változás elfogadása, a túl részletes, túl korai követelményspecifikáció elhagyása nem jelenti azt, hogy a fejlesztés tervezetlenül és gyenge minőségű kódot eredményezve folyjon. Az agilis csapatok számára is elengedhetetlen a tervezés, mindössze a tervezés időhorizontja és a távoli elképzelések kidolgozottságának a szintje változik. A sikeres agilis szoftverfejlesztés elengedhetetlen feltétele a rendszeres tervezés és a szigorúan értelmezett, minőségi kód.

> 10. Elengedhetetlen az egyszerűség, vagyis az elvégezetlen munkamennyiség maximalizálásának művészete.

Mivel a szoftverfejlesztés innovatív tevékenység – azaz elméleteket valósítunk meg, ismeretlen utakon járunk –, nagyon nehéz a fejlesztés elején megmondani azt, hogy melyik elképzelésünk lesz valóban működőképes, és melyik bukik el a gyakorlati alkalmazás során. Éppen ezért törekednünk kell arra, hogy csak olyan problémákat oldjunk meg, amelyeket jól értünk. Tipikus ellenpélda az általános felhasználású, paraméterezhető kódrészletek fejlesztése. A gyakorlatban ezek nagy részét mégsem sikerül felkészíteni azokra az esetekre, amelyekkel később találkozni fogunk és a komplex kódok csak arra lesznek jók, hogy nagyobb kódmennyiséget kelljen újraírni…

> 11. A legjobb architektúrák, követelmények és rendszertervek az önszerveződő csapatoktól származnak.

A 11. agilis alapelv mögött az a megfigyelés húzódik meg, hogy a komplex környezetekben – és a szoftverfejlesztés ilyen – jelentkező problémákat azok tudják a legjobban megoldani, akik azzal a legközelebbről találkoznak. Mivel minden probléma más, ezért a csapat tagjainak együttes tudásával orvosolhatók a legsikeresebben úgy, hogy a csapat saját maga szervezi meg a megoldás kialakításának módját. Ezt kívülről (akár felülről) jövő emberek sokkal nehezebben tudják megtenni.

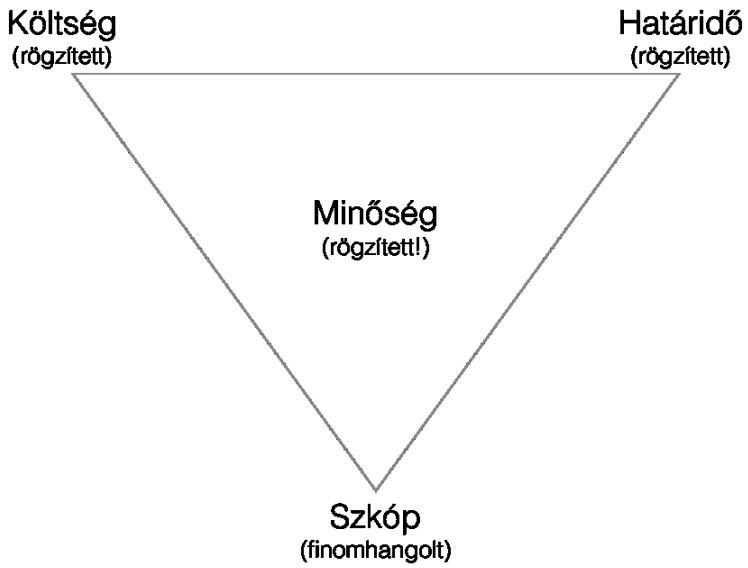
> 12. A csapat rendszeresen mérlegeli, hogy miképpen lehet fokozni a hatékonyságot, ehhez hangolja és igazítja a működését.

A szoftverfejlesztés minden esetben tanulási folyamat. A projekt előrehaladásával a csapat tagjai egyre jobban értik az üzleti problémát, egyre gyakorlottabbak az alkalmazott technológiák használatában, egyre jobban megismerik egymást, egyszóval a csapat is folyamatosan fejlődik, változik. Ezeknek az új információknak be kell épülniük a csapat napi rutinjaiba, módszereibe. Ezt a tanulást szolgálja a rendszeres felülvizsgálat és fejlesztés.

A fordított vasháromszög

Az agilis szoftverfejlesztési szemlélet tehát merőben új alapokra helyezi a szoftverfejlesztési projektek menedzsment módszereit. A folyamatok és módszertanok helyett a működő termékre, a motivált és szorosan együttműködő emberekre helyezi a hangsúlyt.

Ennek megfelelően a projektmenedzsment vasháromszögét is érdemes fordított helyzetben értelmezni (1-5 ábra).



1-5 ábra: Fordított vasháromszög

Az agilis fejlesztésben a szkóp az egyetlen, amit nem ismerünk biztosan. Ez nem azt jelenti, hogy nem tudjuk, hogy mit akarunk fejleszteni, a gyakorlatban inkább arról van szó, hogy rögzítettük az üzleti problémát, amit meg akarunk oldani, de nyitva hagyjuk a megoldás pontos módját.

Az önszerveződő csapat feladata az, hogy találjon olyan megoldásokat a problémára, amelyek megvalósíthatók a rögzített határidőre és a rögzített költségkereten belül.

A projektet a legtöbb agilis módszer rögzített időtartamú iterációkra bontja fel, melyeken állandó összetételű csapat dolgozik, tehát iterációnként mindenképpen rögzített a határidő és a költségkeret is.

A csapattal szembeni legfontosabb elvárás, hogy minden iteráció végén tesztelt, működő, az elvárásoknak megfelelően dokumentált terméket szállítson le. A minőség tehát a költség és határidő kényszerekkel együtt szigorúan rögzített paramétere a projektnek.

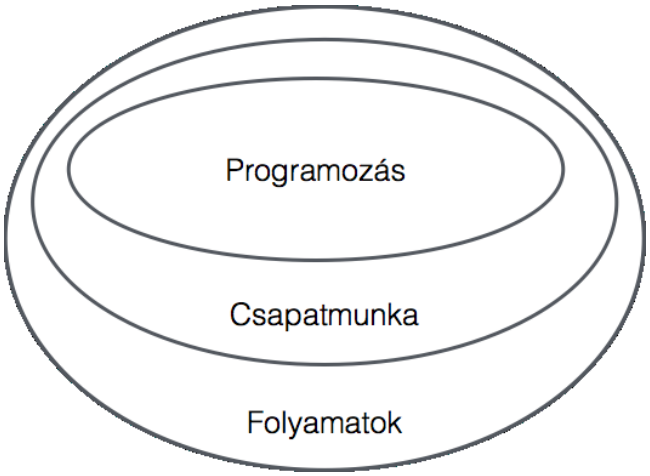
A továbbiakban a legelterjedtebb agilis módszereket fogjuk röviden bemutatni.

Elterjedt agilis módszerek

A fentiekből látszódik, hogy az agilis szoftverfejlesztés egyfajta szemléletmód vagy kultúra, nem pedig egy szigorúan meghatározott módszertan. Olyannyira nem, hogy még az agilis szemléletet megvalósító gyakorlati módszereket sem célszerű módszertannak nevezni. Már csak azért sem, mert mindegyikben közös, hogy a gyakorlati módszerek kialakítását az adott problémát megvalósító csapatra bízzák. Ezek a módszerek elsősorban olyan keretet adnak a csapatok kezébe, amelyeken belül szabadon alakíthatják a konkrét megvalósítási gyakorlatot. Helyesebb tehát ezeket keretrendszernek nevezni.

Extreme Programming (XP)

Az *Extreme Programming*, rövidebb nevén XP első alkalmazása 1996-ig nyúlik vissza. Az XP három egymásra épülő réteget határoz meg, melyek a fejlesztési projekteket meghatározzák. Ezek a *programozás*, a *csapatmunka* és a *folyamatok* (1-6 ábra). Az XP által definiált programozási gyakorlatok a mai napig elengedhetetlen részét képezik bármilyen agilis keretrendszernek.



1-6 ábra: Az eXtreme Programming rétegei

Programozás

Az XP fejlesztők inkrementálisan készítik a kódot tesztvezérelt megközelítéssel: kevés egységteszt (*unit test*), majd éppen elegendő kód ahhoz, hogy az egységteszt sikeresen lefusson. Minden kód részlethez létezik legalább egy teszteset, ami igazolja annak helyességét. Attól, hogy egy kód éppenséggel működik, még nem tekintjük késznek! Az XP arra törekszik, hogy az egész rendszert a lehető legrugalmasabbá tegye a változásra azáltal, hogy a lérező legegyszerűbb felépítésre törekszik. Ennek érdekében gyakran refaktoráljuk a kódot.

Csapatmunka

A készülő kód csapatmunka eredménye, ennek megfelelően a teljes fejlesztési munkát a csapatban történő együttműködésre hegyezi ki. Az XP csapatmunkára vonatkozó szabályai kiterjednek a páros programozás alkalmazására, folyamatos integrációra, a túlóra kezelésére, a közös munkaterület (iroda) kialakítására, a szoftverváltozatok kiadására és a kódolási konvenciókra.

Folyamat

Az XP folyamatok a megrendelővel való együttműködésre, a tesztelési és elfogadási tesztek kialakítására, valamint a közös tervezésre koncentrálnak. Az XP úgynevezett *on-site customer* szerepkört definiál, aki a megrendelő oldalát képviseli, és a fejlesztő csapattal napi szintű együttműködésben dolgozik. A felhasználói elfogadási tesztek készítése ennek a szerepkörnek a feladatai közé tartozik. A csapat előtt álló munka mennyiségének becslése a csapat közös feladata, melyre az ún. *planning game* szolgál.

XP alapértékek

Az XP az előzőekben felsorolt összes alapelvet tökéletesen megvalósítja és az alábbi értékekre koncentrál:

- **Kommunikáció.** Mindenki a csapat tagja és napi rendszerességgel, személyesen kommunikálunk egymással. Közösen dolgozunk mindenen a követelményektől a program kódig. A problémára a tőlünk telhető legjobb megoldást igyekszünk biztosítani.
- **Egyszerűség.** Mindent megteszünk, ami szükséges, vagy amit kérnek tőlünk, de semmivel sem többet! Ezzel maximalizáljuk az adott idő alatt elérhető eredményt. A cél felé kis lépésekben haladunk, és a hibáinkat a felmerülés pillanatában, késlekedés nélkül javítjuk. Büszkék leszünk arra, amit készítünk, és ésszerű költségeken belül maradunk.
- **Visszacsatolás.** Minden iterációra tett vállalásunkat komolyan vesszük, és ezt működő szoftver leszállításával bizonyítjuk. Az elkészült terméket hamar és gyakran bemutatjuk, majd figyelmesen meghallgatjuk a visszajelzéseket, és elvégezzük a szükséges módosításokat. A figyelmünk középpontjában a projekt áll, és ehhez igazítjuk a folyamatainkat, nem pedig fordítva.

- **Tisztelet.** Minden eredmény a csapat közös munkájának eredménye, ennek megfelelően egyenlőnek tekintjük egymást és kölcsönös tiszteletet mutatunk a csapattársaink iránt. A fejlesztők tisztelik a megrendelők üzleti tudását, és ugyanilyen tiszteletet kapnak a saját hozzájárulásukért cserébe a megrendelőtől. A menedzsment tiszteletben tartja a jogunkat a felelősségvállaláshoz, és biztosítja az ennek megfelelő hatáskört.
- **Bátorság, kockázatvállalás.** Mindig őszinték vagyunk az előrehaladás és a becslések meghatározásakor. Nem foglalkozunk azzal, hogy előre kifogásokat keresünk a sikertelenségre, mert a sikerre tervezünk. Nem félünk a kihívásoktól, mert senki sincs egyedül. Alkalmazkodunk, valahányszor a változások ezt szükségessé teszik.

Scrum

A Scrum a legelterjedtebb agilis keretrendszer. Az XP-vel szemben egyáltalán nem határoz meg programozási szabályokat, lényegében egy iparágaktól független termékfejlesztési modell. Ennek köszönhetően egyre elterjedtebb a szoftverfejlesztésen kívül is. Elsőként 1995-ben publikálták, de alkotói már 1993-ban alkalmazni kezdték.

Mivel e könyv további fejezetei a Scrum keretrendszert részletesen bemutatják, ezért most csak a legfontosabb információkat foglaljuk össze.

A Scrum gyökerei a két japán kutató által jegyzett „*The New Product Development Game*" (Az új termékfejlesztési játszma) című publikációig nyúlnak vissza. Ebben a szerzők új elektronikai és gépgyártási termékfejlesztési projekteket vizsgáltak. A Scrum lényegében ezeknek a módszereknek a szoftverfejlesztési projektekben továbbfejlesztett változata, mely lassan visszaáramlik az eredeti elektronikai és gépgyártási iparágakba is. A Scrum Guide három alappillérre épül.

- **Átláthatóság.** A megvalósítási folyamat minden lényeges eleme látható legyen mindenki számára, akik a megvalósításért felelősséget vállalnak.
- **Felülvizsgálat.** A Scrum csapatoknak rendszeresen felül kell vizsgálniuk a folyamataikat, eszközeiket és az elkészült terméket azért, hogy a céloktól való eltérést azonnal felismerjék.
- **Adaptáció.** Ha a vizsgálat során bármilyen eltérést tapasztalnak a kívánt állapothoz képest, azonnal módosításokat kell végrehajtaniuk, hogy a projekt továbbra is a cél felé haladjon.

A Scrum a termékfejlesztést úgynevezett *sprint*ekben valósítja meg. Minden sprint egységnyi (állandó) időtartamú fejlesztési időszak, ami nem lehet több egy hónapnál. A fejlesztőcsapatnak minden sprint végén egy működő „termékbővítményt" kell letennie, ami az eddig elkészült funkcionalitásba integrált új képességgel (képességekkel) bír.

A Scrum mindössze három szerepkört definiál. Az első szerepkör a *fejlesztő*. Mindenki, aki a termék előállításában részt vesz, fejlesztőnek számít függetlenül attól, hogy milyen szakmai specializációval rendelkezik. A fejlesztői csapat minden olyan szakértelemmel rendelkezik, ami ahhoz szükséges, hogy működő terméket szállítson le.

A *product owner* (ennek talán a „termékgazda" a helyes magyar fordítása, de a szakma egyelőre nem használja szívesen ezt a magyar megnevezést) szerepkörben lévő csapattag felel azért, hogy a termékkel szembeni elvárásokat felmérje, és azokat egy sorba rendezett listába, az ún. *product backlog*ba rögzítse.

A product backlog egy sorba rendezett lista mindarról, amit a fejlesztő csapatnak el kell végeznie, vagy meg kell valósítania azért, hogy a termék elkészüljön. A lista tetején lévő elem készül el elsőként.

A Scrum csapat a product backlogból minden sprint során megvalósít annyit, amennyit a meghatározott minőségi elvárásoknak megfelelően képes elkészíteni. Minden sprintet egy sprinttervezés előz meg, melynek során a csapat elkészíti azt a tervet, amit a sprint során meg fog valósítani.

A sprint közben a csapat minden tagja, minden munkanapon egyezteti az előttük álló feladatokat, illetve az elért eredményeket. Ezt *napi scrum megbeszélés*nek nevezzük.

A sprint végén a csapat bemutatja az elkészült terméket a projekt egyéb érintettjeinek, illetve értékeli azt, valamint megvizsgálja saját belső folyamatait és munkamódszereit.

A Scrum szabályainak betartásáért és a csapat hatékony együttműködéséért a *scrum master* szerepkörben lévő csapattag felel.

Lean szoftverfejlesztés és a Kanban módszer

A *lean* szemlélet a Toyota által alkalmazott gyártás- és vállalatirányítási módszer, melynek kulcsszerepe volt abban, hogy a Toyota a kétezres évek elejére a világ legnagyobb és egyben legnyereségesebb autógyára legyen. Ennek a szemléletnek a részletes bemutatása több kötetet igényelne, ezért itt csak a legfontosabb alapelveit ismertetjük.

A lean szemlélet következő öt alapelve (1-7 ábra) definiálja azt a folyamatot, amelyen keresztül megvalósítható a hatékony, veszteségmentes értékteremtés.

1-7 ábra: Lean alapelvek

- **Azonosítsd az értéket!** Határozzuk meg, hogy mik azok a legfontosabb szolgáltatások, termékjellemzők, amelyek a termék végfelhasználói számára értéket jelentenek!
- **Térképezd fel az értékfolyamatot!** Határozzuk meg azokat a lépéseket, amelyeken keresztül az értéket teremtő eredmény elkészül, majd a végfelhasználóhoz eljut! Azonosítsuk és szüntessük meg azokat a tevékenységeket, melyek nem járulnak hozzá a felhasználó számára értékes eredmények előállításához!
- **Hozz létre áramlást!** Helyezzük az értékteremtő lépéseket szorosan egymás mellé azért, hogy az érték a lehető legrövidebb időn belül, megszakítás nélkül jusson el a végfelhasználóhoz!
- **Alkalmazz húzóelvet!** Az áramlás beindítása után alakítsuk úgy a folyamatot, hogy az egyes feldolgozási lépéseket az őket követő feldolgozási lépés igényei vezéreljék!
- **Törekedj a tökéletességre!** Vizsgáljuk meg újra és újra a folyamatainkat, és keressünk olyan tényezőket, melyek a fenti alapelvek megvalósítását gátolják! Minden ilyen tényezőt iktassunk ki a folyamatainkból egészen addig, amíg el nem érjük a tökéletes értékáramlást, azaz azt a folyamatot, ahol a felhasználói érték teremtése veszteségmentesen zajlik!

Veszteségek

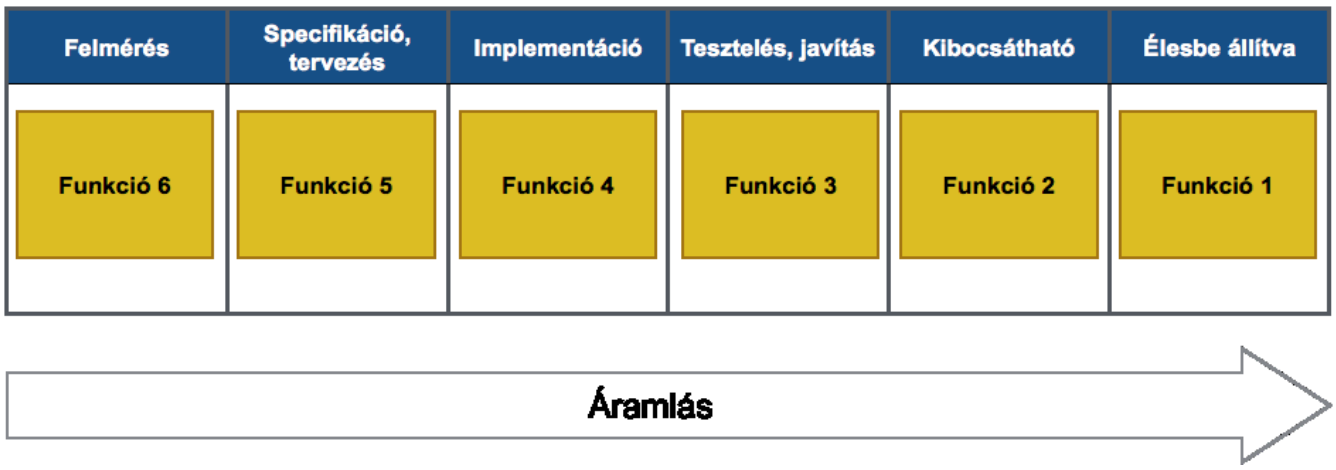
A tökéletes értékáramlás jellemzője, hogy veszteségmentesen zajlik. Ez a veszteségmentesség biztosítja az elérhető legjobb hatékonyságot. Ha tehát a tökéletes fejlesztési folyamatot keressük, akkor fel kell ismernünk és meg kell szüntetnünk a szoftverfejlesztési munka jellemző veszteségforrásait.

- **Részben elkészített munkák.** A részben elkészített munkák olyan feladatok, melyek végrehajtását már megkezdtük, de még nem fejeztük be, így még nem teremtettek értéket a végfelhasználók számára. A túl sok folyamatban lévő munka egyrészt szétszórja a fejlesztői erőforrásokat, másrészt késlelteti az eredmények felhasználóhoz való továbbítását. További kockázatot jelent, hogy a túl sok folyamatban lévő feladat közül néhány jó eséllyel végleg befejezetlen marad, és ezek így tiszta veszteséget jelentenek.

- **Felesleges funkciók.** Olyan funkciók, amelyeket ugyan elkészítettünk és élesbe is állítottunk, de a felhasználók nem használják. Ez jellemzően a megváltozott vagy rosszul felmért követelményeknek, illetve a túl bonyolult megvalósításnak az eredménye.
- **Újratanulás.** Az újratanulás akkor jelentkezik, amikor egy már kitalált vagy ismert megoldást a csapat „elfelejt". Jellemzően a rosszul szervezett, dokumentálatlan kódra, a tervezési döntések dokumentációjának hiányára, illetve a specifikáció és az implementáció közötti túl sok időre, illetve túl sok közbenső szereplő jelenlétére vezethető vissza.
- **Információ tologatás.** Az információ tologatás jellemzően az a veszteség, ami az igény vagy probléma keletkezési helyétől a megoldás leszállítását végző emberekig megtett útból fakad. Ezt az utat nehezítheti a túl sok továbbító személy, illetve a feleslegesen alkalmazott dokumentáció. Gondoljunk csak a vízesés modellben a követelmény felmérését végző üzleti elemzőtől, a tervezést végző architektúra szakértőn át a fejlesztőig és tesztelőig megtett számos információátadási lépésre! Ők sokszor alig találkoznak egymással, formális dokumentumokon keresztül kommunikálnak.
- **Késések, várakozás.** Bármilyen jelenség, ami lassítja az értéket teremtő funkció élesbe állítását a felhasználó számára, veszteségnek tekinthető. Hosszú jóváhagyási idők, a fejlesztők várakozása a specifikációra, a tesztelők várakozása a tesztelhető kódra, a termék kiadás várakozása a tesztelésre és hibajavításra: ez mind veszteség.
- **Feladatváltás.** A feladatváltás általában a túl sok folyamatban lévő munka eredménye és rendkívül negatívan befolyásolja a hatékonyságot. A gyakori feladatváltás gyakori újratanulást, kontextus váltást és stresszt eredményez.
- **Hibák.** A rendszerben hagyott hibák negatív hatását nem kell különösebben magyarázni. Amellett, hogy a felhasználót meggátolják abban, hogy élvezhesse a rendszer előnyeit; adatvesztéstől kezdve a felhasználó folyamatainak megakadását okozhatja, a fent felsorolt veszteségek mindegyikét előidézheti a fejlesztési folyamatban. Az azonnal javítandó hibák növekvő feladatmennyiséget, feladatváltást, újratanulást és késéseket visznek a folyamatba. Éppen ezért a hibamentes termék előállítása már az első kiadáskor alapvető eleme a lean szoftverfejlesztésnek.

Az egydarabos áramlás

Az egydarabos áramlás (1-8 ábra) az az ideális folyamat, amikor minden feldolgozási lépésben pontosan egy munkadarab van. Belátható, hogy ez a folyamat eredményezi a fenti veszteségek leghatékonyabb elkerülési módját.



1-8 ábra: Az egydarabos áramlás

Az ábrán egy olyan ideális áramlást ábrázoltunk, ahol a fejlesztés minden munkafázisában pontosan egy funkció tartózkodik. Az ilyen ideális világban pontosan annyi ideig tart felmérni egy új funkcióval szemben támasztott igényt, mint amennyi ideig lefejleszteni az előzőleg már felmért funkciót, vagy élesbe állítani a néhány lépéssel korábban elkészítettet. Ha sikerülne megvalósítani egy ilyen fejlesztési folyamatot, akkor az

egy meghatározott ritmus szerint folyamatosan képes lenne élesbe állítani egy új termék funkciót. Ez lenne az ideális, veszteségmentes folyamat. Habár ez az ideális folyamat tökéletesen nem megvalósítható, meglepően közel lehet hozzá kerülni. Több olyan szoftverfejlesztő cég is létezik, amelyik ezzel a módszerrel kétheti, vagy akár heti kibocsátási ritmust is képes megvalósítani.

A Kanban módszer

A lean szemlélet az összes agilis fejlesztési modellt áthatja, így a vízesés modellhez viszonyítva bármelyik agilis módszer „lean"-nek tekinthető, ugyanakkor megjelent az agilis szoftverfejlesztés ernyőjén belül egy kifejezetten erre a szemléletre épülő módszer, a Kanban módszer. Ha azt mondtuk, hogy az agilis szemlélet nem egy egységes módszertan, akkor ezen belül a lean szemléletre alapuló módszerekre is kijelenthetjük ugyanezt.

A Kanban módszer a lehető legkevesebb előírást határozza meg a fejlesztő csapatok számára, és – ellentétben az XP és Scrum keretrendszerekkel – nem foglalkozik a fejlesztői csapatban létrehozandó szerepkörökkel. A Kanban legfontosabb előírásai a következők:

- **Tegyük láthatóvá a munkafolyamatainkat!** Használjunk fizikai vagy virtuális táblát, melyen minden munkafázishoz egy önálló oszlop tartozik! Minden egyes munkát (pl. új képesség) írjunk fel egy kártyára és ezeket mozgassuk a táblán annak megfelelően, hogy az adott feladat melyik munkafázisban tartózkodik!
- **Korlátozzuk a folyamatban lévő munkák (*work in progress*) számát!** Minden munkafázishoz rendeljünk egy korlátot, ami azt határozza meg, hogy mennyi munka tartózkodhat egy adott időpillanatban az adott munkafázisban! Ezt a számot általában WIP korlátnak nevezik az angol „work in progress" kifejezés rövidítéséből. Ideális esetben ez a szám egy.
- **Mérjük az átfutási időt!** Mérjük, hogy egy kártya átlagosan mennyi idő alatt halad végig a táblán, és folyamatosan hangoljuk a WIP korlátokat, illetve a munkamódszereinket úgy, hogy ezt az időt a lehető legalacsonyabbra csökkentsük!

A Kanban módszer ebből a néhány alapelvből kiindulva igyekszik növelni a fejlesztési munka hatékonyságát. Sokan igyekeznek szembeállítani a Kanban és a Scrum keretrendszereket részben eltérő szemléletmódjuk miatt, de ez nem helyes megközelítés, a Scrum és a Kanban kiválóan kiegészítik egymást. Általában a Scrumot olyan környezetben alkalmazzák, ahol viszonylag kiszámíthatóak a feladatok (pl. új termék fejlesztése), míg a Kanban modell a kevésbé tervezhető munkakörnyezetekben (pl. támogatás, szoftver üzemeltetés) elterjedtebb. A két modell összehasonlítása iránt érdeklődőknek a **`http://www.adaptiveconsulting.hu/sites/default/files/KanbanEsScrum_MindkettobolALegjobbat_1.pdf`** elektronikus könyv szolgálhat további információkkal.

Összegzés

Ahogyan azt láthattuk, az agilis szoftverfejlesztés több izgalmas irányzat összessége, és nem egy konkrét módszertan. Mindegyik keretrendszer a személyes kommunikációra, a csapatok felelősségvállalására, a változás elfogadására és az intenzív kommunikációra épít.

A következő fejezet a legelterjedtebb keretrendszer, a Scrum bemutatásával foglalkozik.

2. Scrum alapok

Ebben a fejezetben az alábbi témákat ismerheted meg:

- Mi a Scrum?
- Hogyan épül fel egy Scrum csapat?
- Milyen folyamatokat és eszközöket ír elő a Scrum?

Az előző fejezetben megismerhetted az agilis termékfejlesztési szemlélet alapelveit és a legfontosabb módszertani keretrendszereket, amelyek megvalósítják ezeket. Ebben a fejezetben a Scrum bemutatásával folytatjuk az agilis világgal való ismerkedést. Egy felmérés szerint az agilis szoftverfejlesztést alkalmazók mintegy kétharmada nyilatkozott úgy, hogy Scrum vagy valamilyen Scrum/XP hibrid módszert alkalmaz. Ezzel magasan a Scrum a legelterjedtebb modell. A következőkben a Scrum alapjait lefektető Scrum Guide 2013-as kiadása alapján vizsgáljuk a Scrum legfontosabb alapelveit.

A Scrum Guide ingyenesen elérhető a **`https://www.scrum.org/Scrum-Guide`** címen.

Mi a Scrum?

A Scrum egy agilis termékfejlesztési keretrendszer, amelynek segítségével a csapatok komplex, nehezen tervezhető problémákat oldhatnak meg, és innovatív megoldásokat szállíthatnak le a lehető legjobb minőségben. A Scrum egy viszonylag egyszerű modellre épülő rendszer, melyet könnyű megérteni, de rendkívül nehéz jól alkalmazni.

A Scrum keretrendszer a Scrum csapatból és az ezeken belüli szerepkörökből, eseményekből, valamint eszközökből és szabályokból áll. Mivel a Scrum egy keretrendszer, ezért nyitva hagyja azokat a gyakorlati munkamódszereket, melyeket a csapatok alkalmazhatnak a sikeres termékfejlesztés érdekében. Ebben a könyvben ugyanakkor igyekszünk néhány gyakorlati módszert is bemutatni.

Ahogyan az előző fejezetben is említettük, a Scrum három alapvető építőköve a következő:

- **Átláthatóság.** A megvalósítási folyamat minden lényeges eleme látható legyen mindenki számára, akik a megvalósításért felelősséget vállalnak.
- **Felülvizsgálat.** A Scrum csapatoknak rendszeresen felül kell vizsgálniuk a folyamataikat, eszközeiket és az elkészült terméket azért, hogy a céloktól való eltérést azonnal felismerjék.
- **Adaptáció.** Ha a vizsgálat során bármilyen eltérést tapasztalnak a kívánt állapothoz képest, azonnal módosításokat kell végrehajtaniuk, hogy a projekt továbbra is a cél felé haladjon.

A Scrum legfontosabb szabálya az adaptivitás, azaz a gyakorlati tapasztalatok gyűjtése, értékelése és az ezek birtokában történő változás, alkalmazkodás.

A Scrum csapat

A *Scrum csapat* résztvevői a *product owner*, a *scrum master* és *fejlesztő csapat*. A Scrum csapatok **önszerveződőek**, azaz saját maguk határozzák meg, hogy hogyan érjék el céljaikat, nem pedig külső irányítás alatt dolgoznak. A Scrum csapatok másik sajátossága a **kereszt**

funkcionalitás. Ez azt jelenti, hogy a csapat minden olyan szakértelemmel rendelkezik, ami ahhoz szükséges, hogy rendszeres időközönként működő terméket szállítson. Tehát a csapat szállítási képessége nem függ semmilyen külső szakértőtől vagy szakértelemtől. Egy szoftverfejlesztő Scrum csapat például a következő kompetenciákkal rendelkező emberekből állhat:

- **Szoftverfejlesztők.** Az adott platformhoz értő szoftverfejlesztők, beleértve a front-end, back-end területeket.
- **Üzleti elemzők.** Olyan szakértők, akik a felhasználó igények részletes megértését, specifikálását végzik. Kisebb projektekben ők gyakran a product owner szerepkört látják el.
- **Tesztelők.** A szoftver szisztematikus tesztelését végző szakemberek. Manuális, automatikus teszteket terveznek és hajtanak végre. Néha ők végzik a termék kibocsátásra való összeállítását is.
- **Szakíró (*technical writer*).** A termékhez kapcsolódó dokumentációk professzionális elkészítését végző szakember.
- **UX/UI dizájner.** A felhasználói felületek, felhasználói élmény megtervezését és megvalósítását végző szakember.

Természetesen ez a kompetencia-összetétel projektenként más és más lehet, illetve az is gyakori, hogy egy-egy csapattag egynél több kompetenciával rendelkezik ezek közül. Nagyon fontos, hogy ezek az emberek nem független szerepköröket töltenek be, hanem a termék tervezését és megvalósítását közösen, egymással szoros együttműködésben végzik.

Product owner

A *product owner* felel azért, hogy megfogalmazza azokat a célokat, amelyek eléréséért a Scrum csapat dolgozik. Más szóval, az ő felelőssége a termékvízió és a termék jellemzők megfogalmazása és ezek világos, érthető kommunikációja a fejlesztő csapat felé. Kisebb projektekben gyakran a product owner végzi a termékfunkciók specifikációját is. A product owner egy sorba rendezett listában, a *product backlog*ban vezeti azokat a feladatokat, amiket a fejlesztői csapatnak meg kell oldania. A product owner felelőssége az alábbiakra terjed ki:

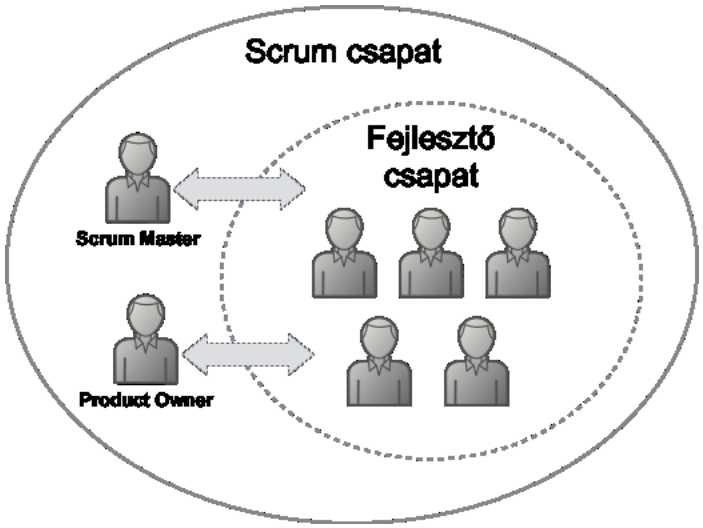
- Product backlog elemek világos megfogalmazása.
- A product backlog elemeinek rendszeres sorba rendezése úgy, hogy azok a lehető legnagyobb értéket termeljék.
- Biztosítania kell, hogy a product backlog a csapat minden tagja számára elérhető és érthető legyen, valamint világosan tükrözze, hogy mik a sorban következő elérendő célok.

Nagyobb projektekben gyakori, hogy a fejlesztő csapat valamelyik tagja (például az üzleti elemző) végzi a fenti feladatok valamelyikét, de a felelősség ebben az esetben is a product owneré. A product owner egyetlen személy, nem egy testület. Függetlenül attól, hogy a termék fejlesztése több érdekelt céljait is szolgálja, ezek megértése és az ezek közötti összhang megteremtése, valamint konkrét célkitűzésekké alakítása a product owner mint egyetlen személy felelőssége.

> *Az egyetlen személyben megtestesülő product owner gyakori támadási pont a Scrummal szemben, mivel a gyakorlatban igen nehéz megfelelő (kompetenciával és szabad kapacitással rendelkező) embert találni erre a pozícióra. A tapasztalatok ugyanakkor azt mutatják, hogy az igazán sikeres termékek mögött minden esetben egy ilyen ember áll.*

Fejlesztő csapat

A *fejlesztő csapat*ot azok a szakemberek alkotják, akik előállítják a terméket. Ahogy azt korábban is tárgyaltuk, a csapat önszerveződő és keresztfunkcionális (2-1 ábra). A Scrum nem különböztet meg pozíciókat a fejlesztő csapaton belül, mindenki fejlesztő. A csapaton belül nincsenek kisebb csapatok, a csapat egyetlen, oszthatatlan egész.



2-1 ábra: Scrum csapat és fejlesztő csapat

A fejlesztő csapat mérete elég kicsi ahhoz, hogy a csapatszellem és az önszerveződés még működjön, és elég nagy ahhoz, hogy képes legyen az adott környezetben rendszeres, maximum 30 napos időközönként működő terméket vagy – ahogy a Scrum Guide fogalmaz – termékbővítményt leszállítani. A fejlesztő csapat maximális mérete 9 fő.

> *A gyakorlati tapasztalatok azt mutatják, hogy 7 fős fejlesztői csapatlétszám felett már erősen jelentkeznek az önszerveződés korlátai, ezért egyre inkább elfogadott az az álláspont, hogy az ideális csapatlétszám 5 fő körül van.*

A scrum master és a product owner akkor tekintendő a fejlesztő csapat tagjának, ha tevékenyen részt vesz a fejlesztésben.

Scrum master

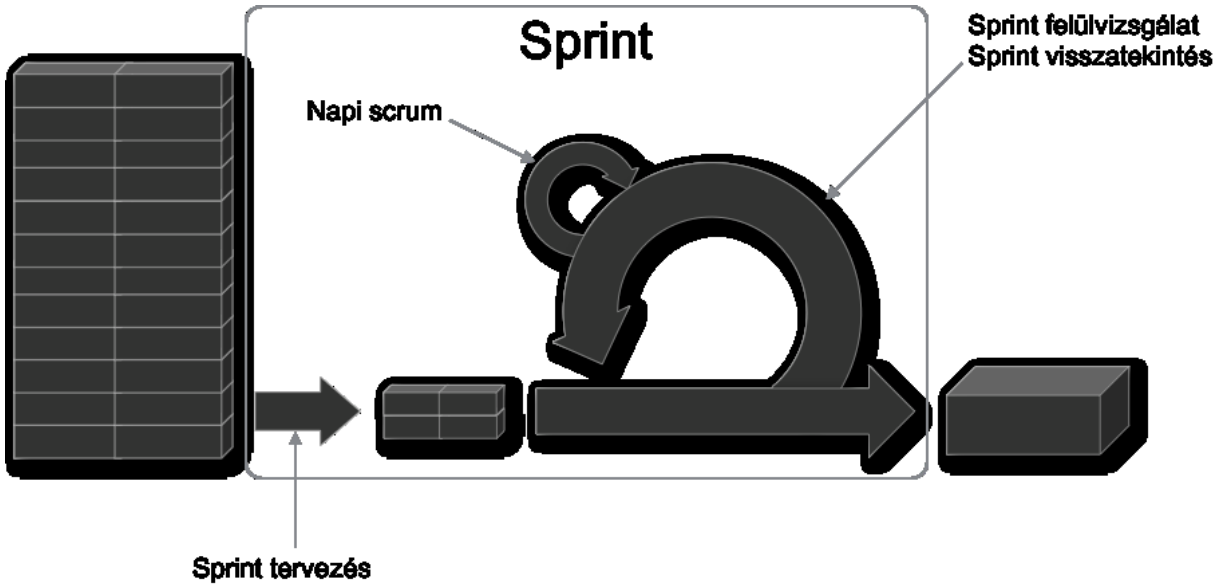
A *scrum master* felel azért, hogy a szervezetben és a csapaton belül mindenki megértse, elfogadja és betartsa a Scrum szabályait. A csapat szemszögéből nézve a scrum master – a Scrum Guide megfogalmazása szerint – a csapatot szolgáló vezető. Ez azt jelenti, hogy a csapatot nem utasításokkal irányítja, hanem biztosítja azt a légkört és szellemiséget, amiben a Scrum csapat hatékonyan tud működni. A csapat és a csapatot körülvevő szervezet felé is képviseli a Scrum alapelveit, figyeli a csapatot érő hatásokat – például vezetői utasítások, céges szabályzatok –, és igyekszik ezeket úgy befolyásolni, hogy azok a Scrum csapat hatékony működését szolgálják.

A scrum master támogatja a product ownert a product backlog vezetésében, a product backlog kommunikálásában a fejlesztő csapat felé, illetve az adaptív környezetben történő tervezés elsajátításában.

A Scrum csapatot segíti az önszerveződés megvalósításában, a keresztfunkcionális csapatmunka szabályainak elsajátításában. A csapat felé egyfajta coachként és moderátorként viselkedik.

Scrum események

A Scrum meghatározza azokat az eseményeket (2-2 ábra), amelyeket a Scrum csapatnak meghatározott időközönként rendszeresen és elhagyhatatlanul meg kell tartania. A Scrum által definiált összes esemény időkeretbe szorított, azaz nem léphetik túl azt az időtartamot, amit a Scrum Guide vagy a csapat meghatároz számukra.



2-2. ábra: Scrum események

A sprint

A sprint a Scrum legfontosabb eseménye. A Scrum csapatok tevékenységét a sprintek foglalják keretbe. Minden további esemény a sprinteken belül zajlik, és elsősorban a tanulást, felülvizsgálatot és a körülményekhez való alkalmazkodóképesség fenntartását szolgálja. A sprint a sprint tervezésből, a fejlesztési munkából, a napi scrum egyeztetésekből, a sprint felülvizsgálatból és a sprint áttekintésből áll.

Egy sprint pontosan az előre meghatározott időpontig tart, annak időtartamát sem rövidíteni, sem növelni nem szabad. Ennek hossza nem haladhatja meg a 30 napot. Hasznos, ha egy csapat előre rögzíti a sprintek időtartamát és minden sprint pontosan ugyanannyi naptári napig tart, hogy segítsen egyfajta csapat ritmus kialakításában.

> *A Scrum csapatok leggyakrabban 2 hetes sprinteket határoznak meg. Ez az időtartam a legtöbb esetben elég ahhoz, hogy a csapat működő termékbővítményt állítson elő.*

A konkrét fejlesztési feladatokat a sprint időtartama alatt végzi el a csapat. Minden sprintnek jól meghatározott célja van, amit a csapat a sprint végére el kíván érni. Azt, hogy a csapat mennyi feladatot vállal be egy sprint időtartamára, kizárólag a csapat tagjai határozhatják meg, azt kívülről erőltetni nem lehet. A sprint elindítását követően:

- Nem módosítható a sprint eredeti terve olyan mértékben, amely veszélyeztetné a sprint céljának elérését.
- A termék és kód minőségére vonatkozó szabályok nem szeghetők meg.
- Előfordulhat, hogy a sprint feladatait tovább pontosítják, kisebb módosításokat hajtanak végre a terven, ha az újonnan felmerült információk ezt indokolják, és ezek egyébként nem veszélyeztetik a sprint kitűzött céljainak elérését.

Ha a sprint közben olyan körülmények merülnek fel, amik előreláthatóan megakadályozzák a csapatot a sprint céljának elérésében, akkor a sprintet le kell állítani és új sprintet kell tervezni és indítani.

Sprint tervezés

A sprinteket a legegyszerűbb maximum 30 napig tartó mini projekteknek felfogni. Mint minden projektet, a sprinteket is előre meg kell tervezni. A Scrum csapatnak világos elképzeléssel kell rendelkeznie arról, hogy ***mit*** kell elérnie, és hogy ezt ***hogyan*** fogja tudni megtenni. A sprint tervezés kollektív munka, tehát a teljes Scrum csapat részt vesz benne.

A Scrum Guide 8 órában maximálja egy sprint tervező megbeszélés hosszát, 30 napos sprintek esetében.

Mit csináljunk?

A sprint tervezés első lépésében meg kell határozni, hogy mi az a funkcionalitás, amit a csapatnak célszerű megvalósítania ebben a sprintben. A tervezés első lépését a product backlog alapján végzi el a csapat. A product owner felelőssége, hogy a product backlogot úgy készítse elő, hogy világosan érthető legyen a backlog elemek tartalma, célja, illetve a backlog elemek olyan sorrendben szerepeljenek, amilyen sorrendben történő megvalósítás a lehető legnagyobb értéket biztosítja a végfelhasználók számára.

> *Rendszeresen tartott 8 órás sprint tervezések gyorsan képesek kiégetni a csapat lelkesedését, ezért célszerű keresni ennek csökkentési lehetőségeit. A Scrum Guide nem rögzíti önálló eseményként a product backlog elemeinek közös átbeszélését és megértését, de a gyakorlat azt mutatja, hogy a sprintek általában akkor sikeresek, ha a csapat tagjai már jóval a sprint tervezése előtt elkezdenek ismerkedni a megoldandó feladattal. Sokan előtervezésnek nevezik ezt a tevékenységet és rendszeres formális találkozóként szervezik meg, hasonlóan a sprint tervezéshez. Heti 1-2 alkalommal tartott ½-1 órás megbeszélések, illetve 2 hetes sprintek alkalmazása esetén egy 5 fős Scrum fejlesztői csapat 2-4 óra alatt képes elkészíteni a sprint tervet.*

A Scrum csapat, miután megértette, hogy mely backlog elemek megvalósításával tudná maximalizálni a felhasználók számára teremtett értéket, megbecsüli, hogy a sprint során milyen kapacitással fog rendelkezni. A kapacitás lényegében az a fejlesztői munkamennyiség, amit a feladatok megoldására képes a csapat fordítani. Ezt befolyásoló tényezők elsősorban az alábbiak lehetnek:

- Csapatlétszám
- Munkanapok és munkaszüneti napok az adott sprintben
- Tervezett szabadságok
- Egyéb, nem sprinthez kötődő elfoglaltságok (például konferenciák, ajánlatírás, továbbképzések stb.)
- Egyéb, valamekkora valószínűséggel bekövetkező nem várt események (például az év elején várhatóan jelentkező influenzajárvány miatti megbetegedések)

Miután a csapat megértette, hogy milyen célokat kell elérnie a sprintben, illetve hogy mekkora a rendelkezésre álló kapacitása, meghatározza a sprint szkópját, vagyis azt, hogy mely backlog elemeket fogja a sprintben megvalósítani. Ennek meghatározásához szüksége van a backlog elemeinek megvalósításához szükséges ráfordítási idő ismeretére. Ezt a megvalósításhoz szükséges ráfordítást a backlog elemek mérete fogja megadni. A backlog elemek méretének becslését a következő fejezetben fogjuk tárgyalni.

A tervezés első lépésénél a product owner szerepe kulcsfontosságú, hiszen a csapat számára ebben a lépésben kell világosan érthetővé tenni, hogy milyen célokat kell elérni.

A tervezés első lépésének eredménye a sprint céljának világos meghatározása.

Hogyan valósítjuk meg?

Az első lépésben a csapat meghatározta a sprint célját, ezen belül azt, hogy mi az a funkcionalitás, amit a sprintben meg fog valósítani. Ehhez megértette a product backlog kiválasztott elemeinek részleteit, méretét, illetve meghatározta azt a kapacitást, ami a rendelkezésére áll a sprint alatt.

A következő lépés a sprint során elérendő célokhoz vezető út megtervezése, a sprint backlog elkészítése. A Scrum Guide definíciója szerint a *sprint backlog* a kiválasztott product backlog elemek és az ezek megvalósítását tartalmazó terv együttesen.

A fejlesztő csapat ilyenkor a rendszer tervezésével foglalkozik, illetve azoknak a feladatoknak az azonosításával, amit el kell végezni annak érdekében, hogy az új funkcionalitást megvalósítsuk. Előfordulhat, hogy a csapat bizonyos feladatokat nyitva hagy, melyek részletes tervezését a sprint közben fogja elvégezni. Erre a csapatnak joga és lehetősége van. A csapatnak azonban rendelkeznie kell maximum egy nap átfutási idő alatt megvalósítható feladatokra lebontott elemekkel, amelyeket a sprint első néhány napján fog megvalósítani azért, hogy a sprint mindenképpen tervezetten induljon.

Ekkor a csapatnak újra lehetősége van megvizsgálni, hogy a tervezés első lépésében kalkulált kapacitás elegendő-e a sprint kitűzött céljainak megvalósítására. Előfordulhat, hogy a tervezés során a fejlesztő csapat olyan problémákat azonosít, amelyekkel nem volt tisztában az első tervezési lépés során, ezért szükségessé válhat a sprint céljainak módosítása.

A második tervezési lépés alatt a product ownernek nem szükséges jelen lennie, mivel ekkorra a megvalósítandó funkciók elvileg már tisztázottak. Fontos ugyanakkor, hogy a product owner legyen elérhető a csapat számára arra az esetre, ha bármilyen olyan új információ vagy kérdés merülne fel, ami miatt az első tervezési lépésben meghatározottak módosítására vagy pontosítására lenne szükség.

A sprint tervezés második lépésének a végén a csapatnak rendelkeznie kell egy elképzeléssel arról, hogy milyen célokat fog a sprint végén elérni, és ezeket nagyjából milyen lépéseken keresztül tervezi megvalósítani. A csapat gyakorlatilag vállalást tesz arra, hogy mit fog a sprint végére elérni.

Fontos, hogy csak a csapat tehet vállalást, azt sem a menedzsment, sem a product owner nem írhatja felül!

Sprint cél

A 2013 közepén megjelent Scrum Guide fokozottabb hangsúlyt fektet a sprint céljának meghatározására. A sprint céljának valami többletet kell hordoznia azonkívül, mint hogy „az összes kiválasztott backlog elemet megvalósítjuk". A jól megfogalmazott sprint cél nyitva hagyja a lehetőséget a csapat előtt, hogy kreatív megoldásokat találjon a cél elérésére, illetve segít megértetni a csapat tagjaival azt az értéket, amit a projekt és a sprint során előállítanak. A sprint célnak elég összetettnek kell lennie ahhoz, hogy csak a csapat együttes munkájával legyen elérhető, hogy a csapat tagjainak együttműködését ösztönözze.

Napi scrum

A napi scrum egy ***maximum 15 perc*** hosszúságú megbeszélés, amelynek célja, hogy a csapat tagjai összehangolják munkájukat annak érdekében, hogy a sprint célt elérjék. A Scrum Guide úgy fogalmaz, hogy a napi scrum egyeztetést 24 óránként kell megtartani. A Scrum Guide nem határozza meg a napszakot, amikor a napi scrum egyeztetést meg kell tartani, de a csapatok többsége a reggeli, munkakezdés elején lévő időpontot választja.

A napi scrum megbeszélést minden nap pontban ugyanakkor, pontban ugyanott kell tartani azért, hogy ne legyen szükség előzetes szervezésre. A napi scrum egyeztetés során a csapat tagjai a következőket osztják meg a csapat többi tagjával:

- Mit csináltam az előző napi scrum megbeszélés óta, ami hozzájárult ahhoz, hogy a csapat elérje a sprint célját?
- Mit fogok tenni a következő napi scrum megbeszélésig, ami továbbviszi a csapatot a sprint céljának elérése felé?
- Látok-e bármilyen olyan akadályt, ami veszélyezteti vagy megakadályozza azt, hogy a csapat elérje a sprint célját?

Ezeken kívül más témáról a napi scrum megbeszélésen nem esik szó. Ha a csapat úgy dönt, hogy egy kérdést részletesen meg kell vitatni, akkor azt a scrum master rögzíti, és megszervezi az ehhez szükséges találkozót.

> *A napi scrum egyeztetések teszik lehetővé a csapat önszerveződő működését az utasításos működési modell helyett, ezért* **a napi scrum egyeztetések elhagyása a scrum keretrendszer elhagyását jelenti!** *A napi scrum az átláthatóság megteremtésének, valamint a reakcióképesség fenntartásának egyik legfontosabb eszköze, ezért a scrum egyik alapköve.*

A fejlesztő csapat a napi scrum megbeszélések alkalmával ellenőrzi, hogy az eddigi haladás elegendő-e ahhoz, hogy a sprint elérje a célját, illetve meghatározza azt, hogy kinek, mit kell tennie annak érdekében, hogy a sprint cél megvalósuljon.

A napi scrum egyeztetések megtartásáért és moderálásáért – azaz azért, hogy a megbeszélés hasznos, a fenti célokat szolgáló, valamint 15 percnél rövidebb legyen – a scrum master felel. A napi scrum megbeszéléseken kizárólag a fejlesztő csapat tagjai szólalhatnak meg.

Sprint felülvizsgálat

A sprint felülvizsgálatot a Scrum csapat minden sprint végén megtartja. A sprint felülvizsgálatnak az a célja, hogy a résztvevők megvizsgálják, hogy a sprint alatt elkészült termék megfelel-e az előzetes elképzeléseknek, illetve értékeljék a sprint során szerzett új információkat. A sprint felülvizsgálat nem formális státusz értékelés vagy prezentáció, hanem informális megbeszélés. A termékfunkcionalitás bemutatásának célja nem az ítélethirdetés, hanem a tanulságok megfogalmazása és a termékkel kapcsolatos tudásunk bővítése.

A sprint felülvizsgálaton a Scrum csapat tagjain kívül részt vehetnek a projekt egyéb érintettjei is, akiket a product owner hív meg. A sprint felülvizsgálat során a résztvevők áttekintik a product backlog elemeit és megvitatják, hogy szükséges-e módosításokat végezni a product backlogon. A sprint során elkészült eredmények alapján elemzik a csapat sebességét, és előrejelzéseket készítenek az egyes backlog elemek várható elkészülési időpontjáról, az esetlegesen vállalt határidők tarthatóságáról.

A sprint felülvizsgálat tárgya tehát a product backlog, az elkészült termékbővítmények, illetve a határidők. Az eredménye pedig egy aktualizált product backlog. A Scrum Guide egy hónapos sprintek esetén maximum 4 órában határozza meg a sprint felülvizsgálat időtartamát. A többi scrum eseményhez hasonlóan a sprint felülvizsgálat megszervezése és moderálása is a scrum master feladata.

Sprint visszatekintés

A sprint felülvizsgálattal ellentétben a sprint visszatekintés tárgya maga a scrum csapat. A sprint visszatekintés során a csapat a saját működését értékeli, és a saját hatékonyság növelésének lehetőségeit kutatja. A sprint visszatekintést a sprint felülvizsgálat és a következő sprint tervezése közötti időben tartja a csapat. A Scrum Guide egy hónapos sprintek esetén 3 órában maximalizálja a sprint visszatekintésre fordítható időt. A scrum master feladata, hogy az eseményt megszervezze és biztosítsa, hogy a résztvevők értik az esemény célját, valamint moderálja az eseményt.

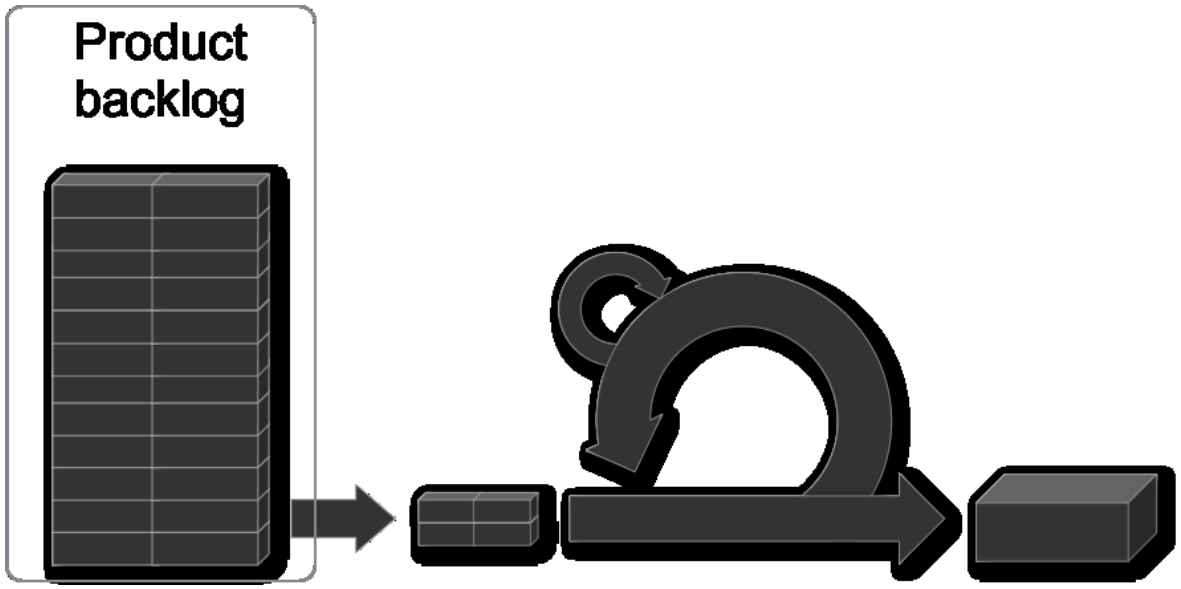
> *A sprint visszatekintés eredménye is egy lista azokról a feladatokról, amelyeket a csapatnak el kell végeznie annak érdekében, hogy hatékonyabb és jobb legyen. A visszatekintés célja nem csupán a hatékonyság növelése, hanem a csapatmorált és a csapattagok hangulatát javító intézkedések azonosítása.*

Szemben a sprint felülvizsgálattal, a sprint visszatekintés szigorúan a csapat belügye. Ha a scrum master nem tagja a fejlesztő csapatnak, akkor szigorúan csak moderátori szerepkörben van jelen ezen a megbeszélésen. A fejlesztő csapat feladata, hogy azonosítsa a saját működésében rejlő fejlődési lehetőségeket.

Scrum eszközök

A product backlog

A product backlog (2-3 ábra) a Scrum fejlesztési projektek legfontosabb eszköze. A product backlogra is igaz a scrum átláthatóságra vonatkozó elvárása, azaz a scrum csapat minden tagja számára elérhetővé kell tenni.



2-3. ábra: Product backlog

A product backlog egy sorba rendezett lista azokról a dolgokról, amiket el kell végezni annak érdekében, hogy a terméket elkészítsük. Bármilyen módosítás vagy fejlesztés, amit a terméken végrehajtunk, kizárólag a product backlogból kerülhet a csapat feladatlistájába. A product backlog vezetése a product owner feladata.

A product backlog egy élő elem, ami soha nem tekinthető késznek, hanem folyamatosan fejlődik, ahogy a scrum csapat egyre több információt és tudást gyűjtött a termékről. Minden esetben tükröznie kell a product owner elképzelését arról, hogy milyen módosításokat, fejlesztéseket tervez végrehajtani a terméken, és ezeket milyen sorrendben látja célszerűnek.

A product backlog vezetése, finomítása egy folyamatos tevékenység, amelyért a product owner felel, de a fejlesztő csapattal együttműködve végzi. A backlog finomítás közben a product owner részletesebben kifejti, szétbontja, törli és összevonja a backlog elemeket. Általános szabályként a backlog finomítása a csapat teljes kapacitásának nem több, mint 10%-át veszi igénybe.

> *A product backlog vezetését a product owner a fejlesztő csapattal közösen végzi, de a Scrum Guide leszögezi, hogy a product backlog elemeinek módosítása, sorrendjének változtatása a product owner saját, önálló joga. Ennek ellenére a jó product ownerek odafigyelnek arra, hogy döntéseiket olyan módon kommunikálják a fejlesztő csapat felé, hogy ők is világosan értsék az azok mögötti szándékot. A csapat motivációjának fenntartásában rendkívül nagy a szerepe a product backlog ésszerű és jó kommunikációval alátámasztott vezetésének!*

A backlog tetején lévő elemek – amelyek időben a legközelebb állnak a megvalósításhoz – általában jobban felbontottak és részletesebben kidolgozottak. Ezek az elemek elegendő információt tartalmaznak ahhoz, hogy a csapat a következő sprint tervezésen felelős döntést hozhasson arról, hogy képes-e megvalósítani a sprint keretein belül.

Sprint backlog

A sprint backlog a sprint szkópját képező backlog elemek és az ezek megvalósítását célzó terv együttese. A sprint backlog legfontosabb szerepe, hogy világosan láthatóvá tegye a csapat tagjai számára, hogy mik azok a feladatok, amiket el kell végezni azért, hogy a csapat elérje a sprint kitűzött célját.

Ennek megfelelően a sprint backlogra fokozottan igaz, hogy a csapat minden tagja számára elérhetőnek kell lennie a sprint ideje alatt.

A sprint backlognak elég részletes tervet kell tartalmaznia ahhoz, hogy a napi scrum megbeszéléseken világosan követhető legyen az előző napi scrum óta eltelt előrehaladás.

Vizuális jelzőeszközök

Az átláthatóság megteremtésének legjobb eszközei az egyértelmű, vizuális visszajelzést biztosító ábrák. A Scrum Guide 2013-as verziójában már nem határozza meg azt, hogy milyenek legyenek ezek az eszközök, de használatukat továbbra is javasolja. A leggyakrabban használt ilyen eszköz a munkahátralék grafikon (*burn-down chart*). Ez a grafikon a sprintben elvégzendő feladatok végrehajtásához szükséges további ráfordítást mutatja a sprintből hátralévő idővel arányosan.

> *A vizuális jelzőeszközökre a könyv későbbi fejezeteiben még visszatérünk a Visual Studio Online képességeinek bemutatásán keresztül.*

A „kész" fogalma

A Scrum Guide teljes tartalmán végigvonul egy fontos fogalom, mégpedig a *„kész"* fogalma (*definition of done*). A „kész" a Scrum csapatok számára egy világosan definiált szempontrendszer, melyhez minden sprintben ragaszkodnak. Az, hogy egy csapat mikor tekint késznek egy product backlog elemet, jelentősen eltérhet csapatról csapatra, ugyanakkor minden csapatnak saját magára nézve szigorú és egységes szempontrendszerrel kell rendelkeznie. A kész legalapvetőbb kritériumai a működőképesség és a hibamentesség. Ezt egészíthetik ki például az automatizált tesztlefedettségre, a kódminőségre, dokumentáltságra vonatkozó kritériumok. A kész fogalmának konzisztens kezelése elengedhetetlen ahhoz, hogy egy Scrum csapat hosszú időn keresztül megbízható teljesítményt nyújtson.

Összegzés

Bemutattuk, hogy a Scrum viszonylag kevés módszertani elemet határoz meg, de azok alkalmazása elengedhetetlen a sikeres Scrum működéshez. A Scrum Guide az elmúlt több mint 10 évben szinte egyáltalán nem bővült. A Scrum alkotói a lehető legtömörebb és legegyszerűbb szabályrendszert igyekeznek meghatározni, azonban ezek mindegyike létfontosságú ahhoz, hogy egy-egy szervezet sikerrel alkalmazza. Ne feledjük: *a Scrumot könnyű megérteni, de rendkívül nehéz alkalmazni!*

A következő fejezetben a Visual Studio Online alapvető szolgáltatásaival ismerkedünk meg.

3. Visual Studio Online – alapok

Ebben a fejezetben az alábbi témákat ismerheted meg:

- Mi is az a Visual Studio Online? Hogyan lehet használatba venni?
- A csapatmunkához használt projektek létrehozása és használatbavétele
- A Visual Studio Online alapvető képességeinek áttekintése
- A Visual Studio fejlesztőkörnyezetének használata, csatlakozás az online szolgáltatásokhoz
- A Visual Studio 2013 csapatmunkát támogató néhány újdonsága

Az előző két fejezetben már megismerkedhettél az agilis módszerek jelentőségével, a termékfejlesztés alapelveivel, és áttekintést kaphattál arról, hogy ezeket az elveket a Scrum módszertan milyen szemléletmóddal valósítja meg. Az ott leírtakból nyilvánvalóan látszik, hogy a csapatmunka támogatásához, a szoftver fejlesztéséhez eszközökre van szükség. A hatékony termékkezelési és fejlesztési módszertan megvalósításához olyan eszközöket célszerű használni, amelyek ezeket a módszereket támogatják.

Az agilis módszereket, a Scrumot, az Extreme Programmingot és a többi megközelítési módot is úgy alkották meg létrehozóik, hogy azok az alkalmazott technológiáktól és szoftver eszközöktől függetlenül tudjanak működni. Ebben a könyvben a Scrumot mutatjuk be részletesen, és amint azt a könyv címéből is láthatod, ezt a .NET technológia környezetében, a Visual Studio és a csapatmunkát támogató Visual Studio Online (korábbi nevén Team Foundation Services) eszközeinek segítségével ismertetjük.

Ebben a fejezetben áttekintést kapsz a Visual Studio Online szolgáltatásról, megtanulod azokat az alapokat, amelyek a következő fejezetekben segítenek a Scrum alapelvek megismerésén túl azok gyakorlati használatában.

A Visual Studio Online

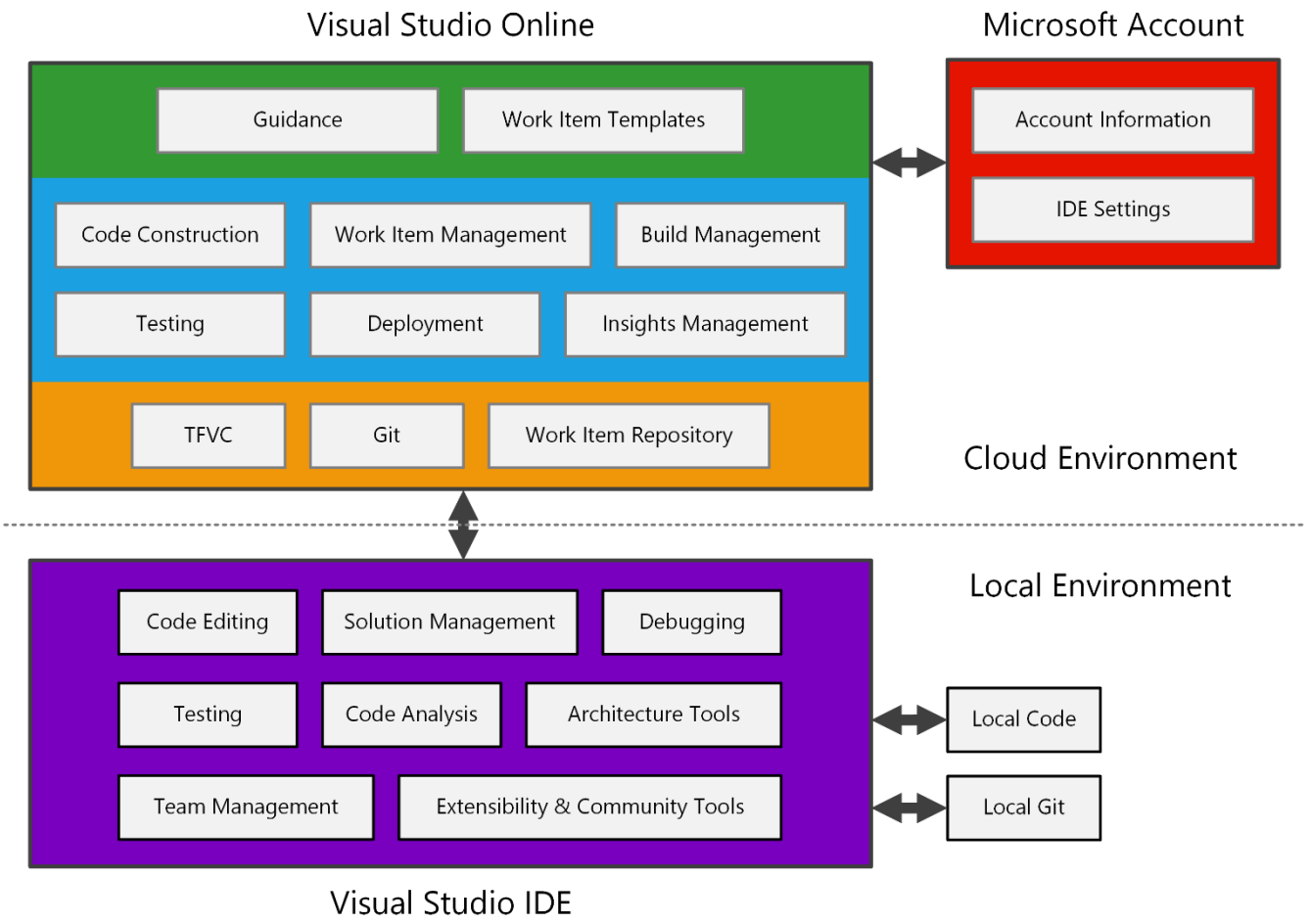
A Visual Studio már a kezdetek (2002) óta a .NET-alapú fejlesztés elsődlegesen használt fejlesztőeszköze, jelenleg a Visual Studio 2013 a fejlesztők által elérhető legfrissebb változat. A szoftverfejlesztés – még akkor is, ha azt csak egyetlen fejlesztő végzi is – a jelen kor kihívásainak megfelelően olyan eszközöket kíván, amelyek hatékony támogatást adnak egy szoftvertermék teljes életciklusának lefedéséhez, az alapötlet kidolgozásától egészen a termék beüzemelésén keresztül annak folyamatos továbbfejlesztéséig.

A Microsoft saját eszköztárában ezt az eszközkészletet két önálló termék integrált együttműködése jelenti már 2005 óta, amikor a Visual Studio 2005-tel párban megjelent a Team Foundation Server, az alkalmazás életciklus menedzsment (*Application Lifecycle Management*, ALM) eszköze. A TFS funkciói 2012-re már a felhőben is elérhetővé váltak, az ingyenesen (korlátozott, öt fős csapatlétszámig) használható Team Foundation Services hamar népszerű lett. A Visual Studio 2013 hivatalos indulásával egy új licenckonstrukció is megjelent, a Visual Studio Online, amely a korábbi dobozos Visual Studio vásárlás mellett szoftverbérlés segítségével – igen kedvező áron – is elérhetővé teszi a fejlesztői környezetet.

A könyv olvasása szempontjából a legfontosabb dolog, hogy a Visual Studio Online ún. Basic változata lehetővé teszi, hogy az itt leírtakat mindenki ingyen próbálhassa ki egy legfeljebb öt tagot tartalmazó csapatban – a csapatmunkához tartozó alapvető funkciók korlátozása nélkül. Természetesen a további változatok már fizetősek (Visual Studio Online Professional és Advanced), ezek az alapvető szoftverfejlesztési technológiákhoz kapcsolódó értéknövelt szolgáltatásokat tartalmaznak.

Ennek a könyvnek nem célja, hogy a termékváltozatok képességeit és a hozzájuk tartozó licenckonstrukciókat részletesen leírja. Ha ezekről többet szeretnél tudni, látogass el az alábbi oldalra: **`http://www.visualstudio.com/products/visual-studio-online-overview-vs`**!

A szolgáltatás legfontosabb elemeit és a közöttük lévő integrációt a 3-1 ábra mutatja be.



3-1 ábra: A Visual Studio Online elemei

Minden csapattag saját számítógépén helyezkedik el a Visual Studio fejlesztői környezete, amely a fejlesztés során használt forráskódot is ugyanerről a számítógépről veszi. Az egyéni munka jelentős részében a szoftverépítést mindegyik fejlesztő ebben a lokális környezetben végzi, mintha csak egyedül lenne.

Természetesen szükség van a többi csapattaggal való együttműködésre. Az ehhez szükséges funkciók a felhőben (a Visual Studio Online esetén a nyilvános felhőben, de egy házon belül használt rendszer esetén akár a vállalati felhőben, adatközpontban) találhatók. A szolgáltatások elérése a Microsoft Accounton keresztül (korábban ez Live ID néven volt ismert) lehetséges. Ez a felhasználói fiók nemcsak az online szolgáltatások elérését biztosítja, de egyúttal a fejlesztőkörnyezet beállításainak tárolására és azok különböző eszközök közötti szinkronizálására is lehetőséget ad.

Az online szolgáltatások kétfajta tárra (*repository*) épülnek. A forráskódokat tároló TFVC (*Team Foundation Version Control*) vagy Git komponensekre, illetve az agilis eljárások alapját jelentő munkadarab gyűjteményre (*Work Item Repository*). Ezeket a tárakat használják a fejlesztői szolgáltatásokat biztosító komponensek, a kódkezeléstől egészen a tesztelésen át a kibocsátások kezeléséig és az élő rendszerek monitorozásáig.

Az alapvető szolgáltatások fölött a módszertan elemeinek, folyamatainak használatát segítő útmutatók (*guidance*) rendszere biztosítja, hogy a csapat az elvárásoknak megfelelően kövesse az együttműködés alapját jelentő – közösen felvállalt – metódust. A *Work Item Templates* modul segítségével a projekt kialakításakor választott metodológia a csapat saját igényeire szabható, elvárásai és projekt-specifikus elképzelései alapján alakítható.

A Visual Studio Online szolgáltatásai nemcsak a Visual Studio IDE-ből vehetők igénybe, hanem akár más fejlesztőkörnyezetekből (pl. Eclipse, IntelliJ IDEA, PhpStorm stb.) is elérhetők. A Visual Studio Online lehetővé teszi, hogy a TFVC mellett a Git forráskódkezelő rendszerét használhassák a fejlesztők.

A csapatmunka előkészítése

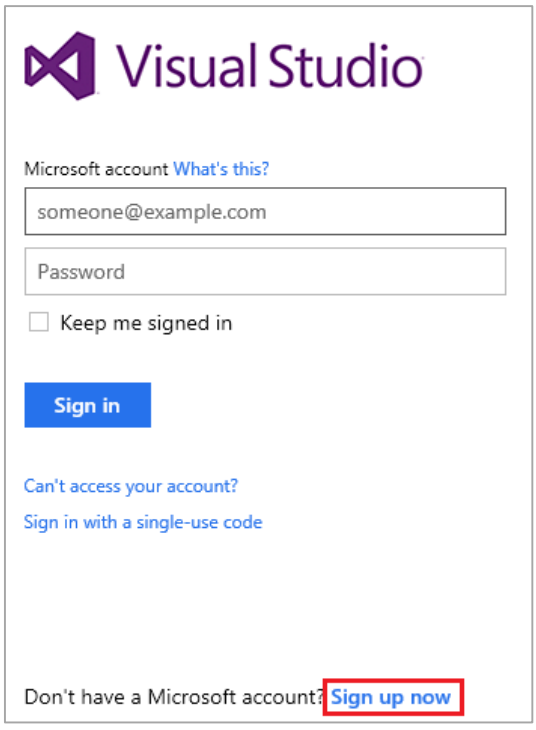
A csapatmunkához értelemszerűen szükség van egy olyan közös tárra, ahol a munkához kapcsolódó információ (forráskód, tevékenységek és egyéb listák, dokumentumok stb.) elérhetők. Ezt a közös tárterületet a Visual Studio Online biztosítja, ahol a közös tevékenység alapjául szolgáló termékhez tartozó összes információt egy project (*team project*) fogja össze. Ahhoz, hogy a csapat tagjai hozzáférjenek ehhez, szükségük van egy olyan Microsoft Account felhasználói fiókra, amelyet a projekt létrehozója felvesz a tagsági listára.

A Microsoft Account létrehozása

Akik Windowst használnak, azok legtöbbjének általában már van Microsoft Accountja (Live ID), van, akinek több is. Ugyanaz a fiók, amit korábban egy SkyDrive vagy Hotmail (Outlook.com) eléréshez hozott létre egy felhasználó, tökéletesen működik a Visual Studio Online-nal is.

Ha véletlenül nem lenne még ilyen felhasználói fiókod, azt könnyen létrehozhatod az alábbi lépések követésével:

1. Látogass el a **`http://www.visualstudio.com/`** weboldalra! Kattints a képernyő tetején található *Get started for free* linkre, és ekkor a bejelentkező képernyőre kerülsz (3-2 ábra), ahol a felhasználói fiókodhoz tartozó bejelentkezési nevedet és jelszavadat adhatod meg.



3-2 ábra: Bejelentkezés

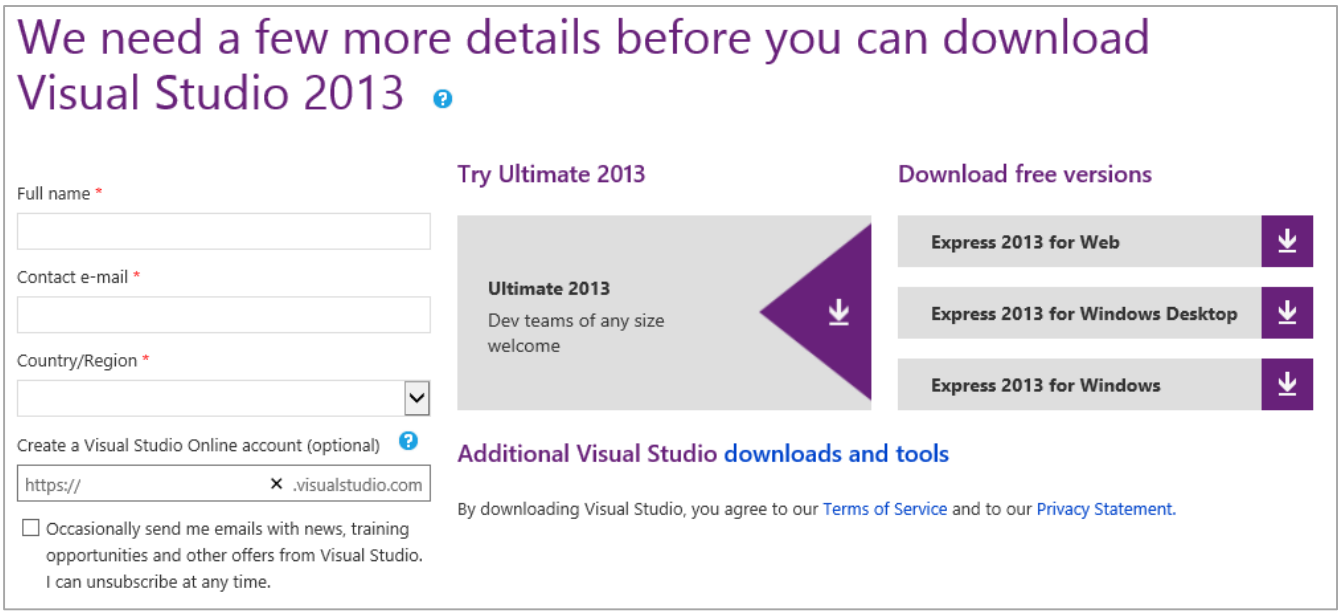
2. Kattints a *Sign up now* linkre! Töltsd ki a megjelenő *Create an account* űrlapot. Felhasználói névnek add meg azt az e-mail címet, amelyhez a felhasználói fiókodat szeretnéd kötni, ez lehet akár egy Gmail, Yahoo! vagy Outlook.com cím is.
3. Töltsd ki a telefonszámodhoz tartozó információt, ez segít abban, hogy a felhasználói információid nagyobb biztonságban legyenek! A különböző biztonsághoz kapcsolódó tevékenységek során a Microsoft Account központja erre a telefonszámra küld neked biztonsági kódokat.

4. A biztonsági kód (CAPTCHA) kitöltése után kattints a Create Account nyomógombra, majd kövesd a fiókod aktiválásához és alapvető beállításaihoz tartozó utasításokat!

A Visual Studio Online fiók létrehozása

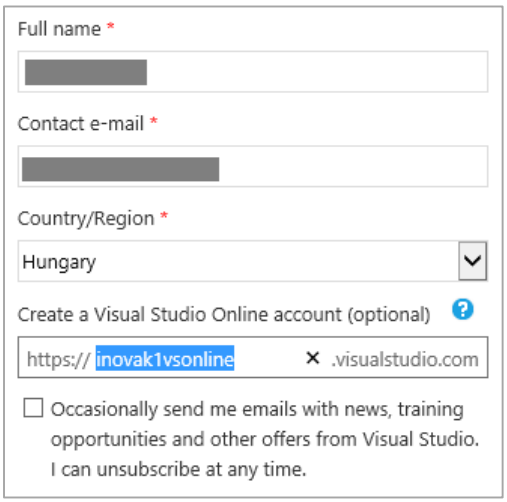
Amennyiben a fenti lépésekkel frissen hoztad létre a felhasználói fiókodat, a folyamat továbbvisz a Visual Studio Online fiók létrehozásához. Ha egy korábban létrehozott fiókhoz szeretnél kapcsolódni, látogass el a **`http://www.visualstudio.com/`** weboldalra, és kattints a képernyő tetején található *Get started for free* linkre, majd jelentkezz be!

A fiók létrehozása néhány további információ megadásával indul, amint azt a 3-3 ábra is mutatja.



3-3 ábra: A Visual Studio Online fiók kiegészítő információinak megadása

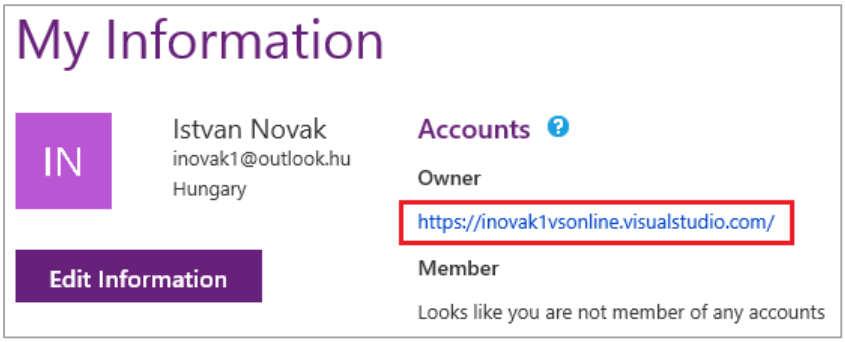
Ezek közül az információk közül a legfontosabb a fiókhoz tartozó URL, amelyet a későbbiekben a saját projektjeid online eléréséhez használhatsz. Ha ezt kitöltötted, eldöntheted, hogy melyik Visual Studio változatot szeretnéd letölteni. Amint valamelyik változatra kattintasz, nemcsak a letöltés indul el, hanem az általad megadott URL is létrejön. A 3-4 ábra az általam mintaként használt felhasználói adatokat jeleníti meg.



3-4 ábra: A mintaként használt fiók URL-je

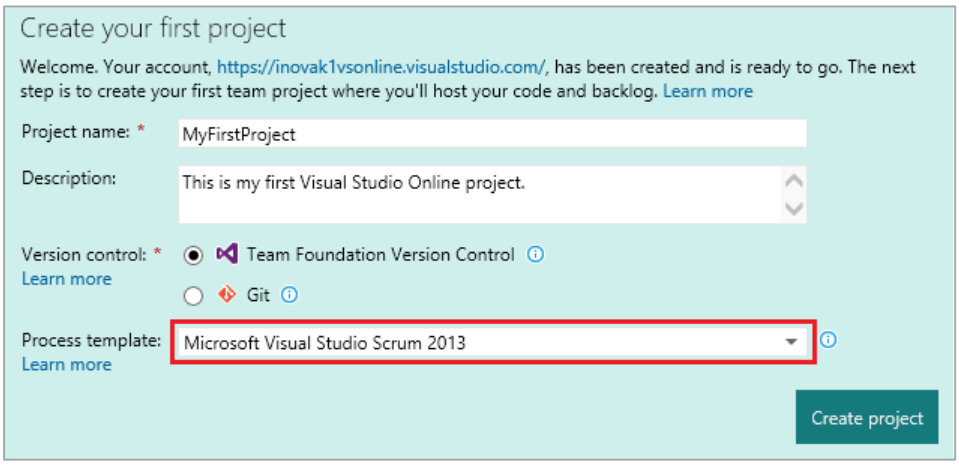
Természetesen ha olyan URL-t adsz meg, amely már létezik – más felhasználó fiókjához kötődik –, a rendszer nem engedélyezi annak elérését, és új URL-t vár tőled. Ha sikerült a fiók létrehozása, a rendszer az ahhoz tartozó profillapra irányít, amint azt a 3-5 ábra is bemutatja.

> *Nem kötelező azonnal letöltened a Visual Studiót, azt később is megteheted, sőt akár több változatot is telepíthetsz. Az Express változatokat szabadon, korlátozás nélkül használhatod. Ha MSDN előfizetéssel rendelkezel, akkor az előfizetésednek megfelelő egyéb fizetős Visual Studio változatot is – az előfizetésben szereplő feltételeknek megfelelően – telepíthetsz számítógépedre. A letöltést a későbbiekben is bármikor elérheted a* ***`http://www.visualstudio.com/`*** *weboldalról.*



3-5 ábra: A Visual Studio Online felhasználói profil

Az *Owner* címke alatt találod a saját fiókodhoz tartozó URL-t, ezen keresztül tudsz hozzáférni a későbbiekben a profilodban található projektekhez. Kattints erre a linkre! Azonnal a saját Visual Studio Online lapodon találod magad. A fiókod még friss, így nem tartozik hozzá egyetlen projekt sem. Erre a szolgáltatás is felhívja a figyelmedet, és néhány adat megadása után – amint azt a 3-6 ábrán láthatod – létre is hoz egy új projektet.



3-6 ábra: Egy új projekt létrehozása

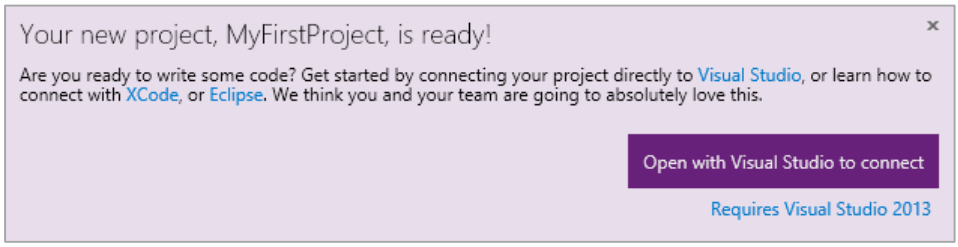
A projekt létrehozásakor meg kell adnod annak nevét (ennek a saját projektjeid között egyedinek kell lennie), annak opcionális leírását, a verziókezeléshez használt tár típusát, illetve a projektedhez használt folyamatsablont.

Ha már korábban használtad a Team Foundation Server verziókezelését, válaszd most is azt! Amennyiben a Git verziótár kezelésében van már tapasztalatod, úgy azt is választhatod – természetesen egy projekten ezek közül csak az egyiket.

A Visual Studio Online – ennek a könyvnek az írásakor – három folyamatsablont kínál fel egy új projekt létrehozásakor. Mivel a könyvünk az agilis módszerekről szól, és azok közül is a Scrumot mutatja be részletesen, válaszd a *Microsoft Visual Studio Scrum 2013* sablont, amely a Scrum aktuális változatának használatához nyújt segítséget!

Kattints a Create Project gombra, és pár perc múlva – amint azt a 3-7 ábrán látható üzenet jelzi – már használhatod is a projektedet!

> *Minden projekthez önállóan választhatod meg a verziókezelés módját és a folyamatsablont is.*



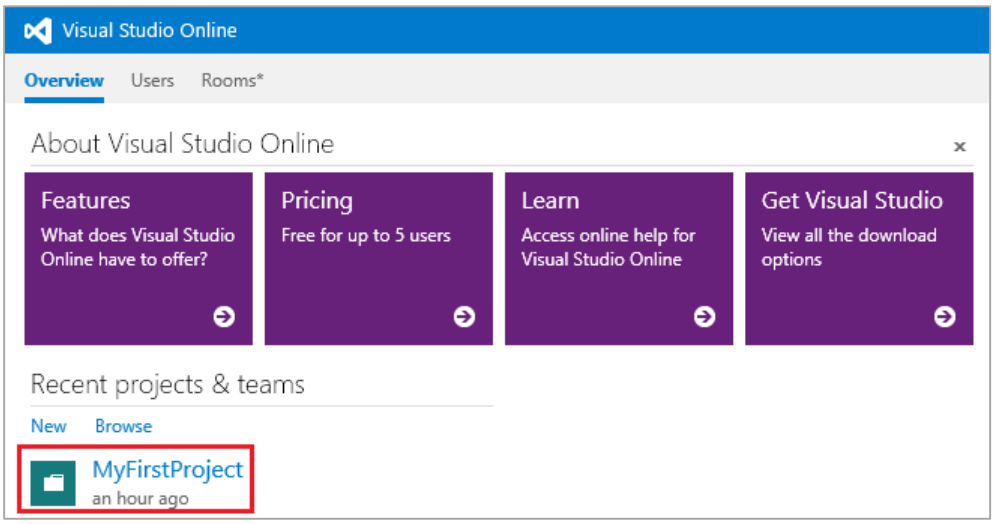
3-7 ábra: A projekt sikeres létrehozását jelző üzenet

A projekt elérése a Visual Studióból

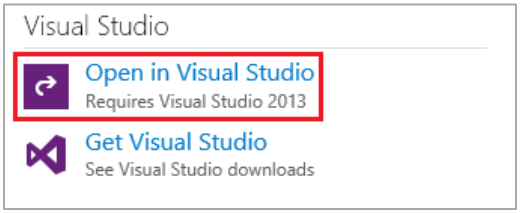
Ha eddig még nem telepítetted a Visual Studiót, itt az ideje! Ha még le sem töltötted, akkor a **`http://www.visualstudio.com/`** oldalon kattints a *Get Visual Studio* linkre, és máris hozzákezdhetsz a letöltéshez, majd a telepítéshez!

A telepítés után először elindítva a Visual Studiót azonnal bejelentkezhetsz az online szolgáltatások eléréséhez használt fiókoddal. Ez azért hasznos dolog, mert így azonnal, újabb felhasználói azonosítás nélkül elérheted projektjeidet.

Ha már bejelentkeztél a Visual Studióba, akkor két lehetőséged is van arra, hogy a korábban létrehozott projektedet elérd. A legegyszerűbb az, ha a Visual Studio Online weblapjáról kiválasztod a korábban létrehozott projektet, amely a *Recent projects & Team* címke alatt található, ahogyan azt a 3-8 ábra mutatja. A projektinformációt tartalmazó lap jobb oldalán található *Open in Visual Studio* link (3-9 ábra) – egy biztonsági megerősítés kérése után – elindítja a telepített Visual Studio 2013 példányt, és automatikusan csatlakozik a beállított projekthez.



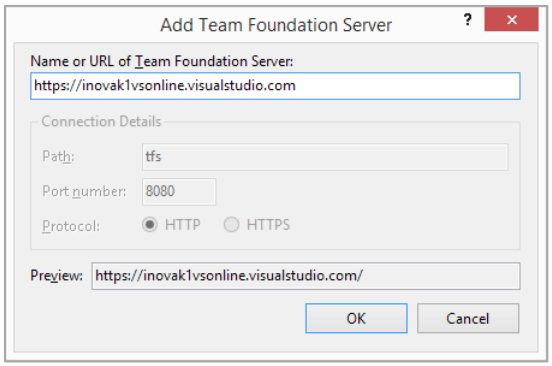
3-8 ábra: a frissen létrehozott projekt kiválasztása



3-9 ábra: Link a Visual Studio indításához

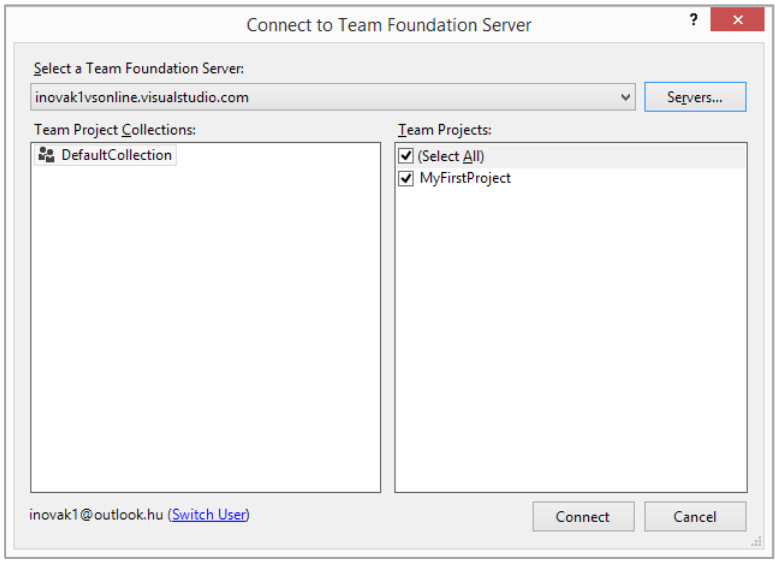
Amennyiben más felhasználói fiókkal jelentkeztél be a Visual Studióba, ez a művelet nem feltétlenül sikerül. Ebben az esetben a Visual Studio belsejéből is könnyen csatlakozhatsz a projektedhez, az alábbi lépéseket követve:

1. A fejlesztőkörnyezet ablakának jobb felső sarkában a felhasználói neved mellett található kis nyíllal megjelenítheted a fiókodhoz tartozó menüfunkciókat. Válaszd ezek közül az *Account Settings*-et! A megjelenő ablak bal oldalán kattints a *Sign out* linkre! A fejlesztői környezet kijelentkeztet. Kattints a *Sign in* gombra, majd add meg a bejelentkezéshez szükséges információt – értelemszerűen a fejezet korábbi részében használt Visual Studio Online fiókhoz kapcsolódót! Zárd le a dialógust a *Close* gombbal!
2. Válaszd ki a *Team* menüből a *Connect to Team Foundation Server* funkciót! A képernyőn megjelenő Team Explorer ablakban kattints a *Select Team Projects* linkre! Megjelenik a *Connect to Team Foundation Server* dialógus, itt kattints a *Servers* gombra!
3. Megnyílik az Add/Remove Team Foundation Server dialógus. Itt kattints az Add gombra! Az Add Team Foundation Server ablakban egyszerűen add meg a Visual Studio Online fiókodhoz tartozó URL-t a HTTPS protokoll használatával, amint azt a 3-10 ábrán láthatod! Én az **`inovak1vsonline.visualstudio.com`** URL-t használtam a folyamat demonstrálására, ennek megfelelően adtam meg a szerver elérését. A csatlakozáshoz kattints az OK gombra!



3-10 ábra: Csatlakozás a szerverhez

4. Zárd le az *Add/Remove Team Foundation Server* dialógust a *Close* gombbal! A *Connect to Team Foundation Server* ablak a jobb oldalon lévő listájában felsorolja a szerveren elérhető projekteket (3-11 ábra). A demonstráció céljából én még csak egyetlen ilyen projektet hoztam létre, ez szerepel az ábrán kiválasztva. Értelemszerűen válaszd itt ki a saját projektedet, majd kattints a *Connect* gombra!

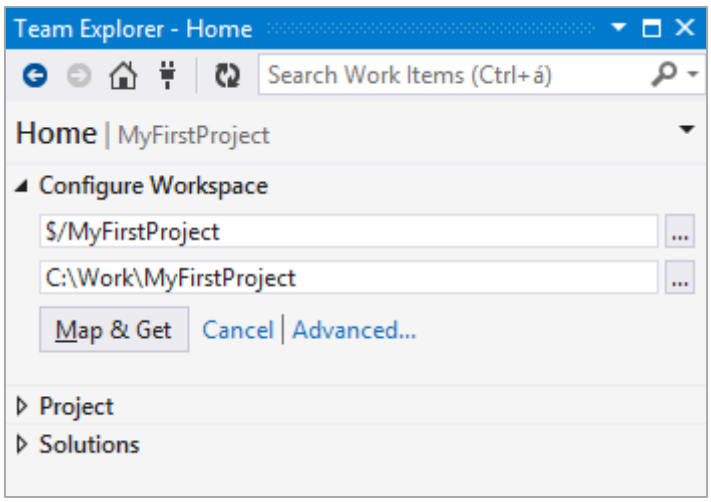


3-11 ábra: A projekt kiválasztása a szerveren

Akár a portálról, akár a Visual Studio környezetéből csatlakoztál a projekthez, még egy fontos lépés hátra van ahhoz, hogy elkezdhess a létrehozott projekttel dolgozni: be kell állítanod a munkaterületed jellemzőit.

> *Attól függően, hogy projekted a Team Foundation Version Control (TFVC) vagy Git tárolási módot használja, a munkaterület beállításának módja különbözik. Itt azzal a feltételezéssel élünk, hogy a TFVC módot használod. A Gitről a C. Mellékletben találsz további hasznos részleteket.*

A beállításhoz kattints a *Configure your workspace* linkre! A Team Explorer megjeleníti a munkaterület beállításait, amint azt a 3-12 ábra mutatja. Az itt található két szövegdoboz közül az első a szerveren lévő projektmappa nevét adja meg (a **`$/MyFirstProject`** mappa a projekt gyökerét jelenti), ezt általában változatlanul kell hagynod. A második mappa a saját számítógépeden lévő lokális könyvtárat adja meg – azt, amelyikbe a projekted fájljait szeretnéd tárolni.



3-12 ábra: A munkaterület beállítása

A beállítások elmentéséhez kattints a *Map & Get* gombra! Ez egyúttal a kiválasztott mappába tölti a projekt szerveren lévő fájljait.

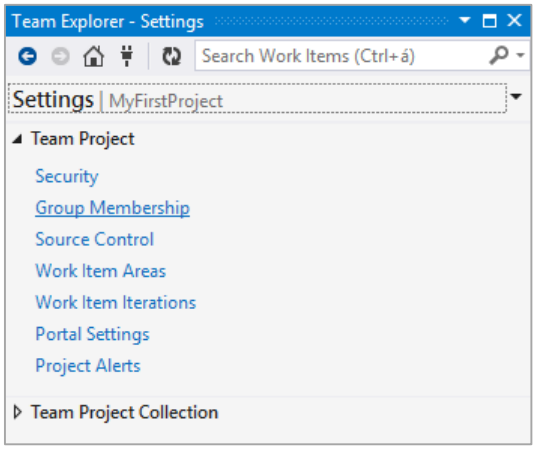
> *A Visual Studio megjegyzi azokat a projekteket, amelyekhez korábban már hozzákapcsolódtál, és ezeket felkínálja a Team Explorer listán. A későbbiekben egyszerűen azokra kell kattintanod a csatlakozáshoz, és nincs szükség arra, hogy a fenti folyamatod újra végrehajtsd.*

Tagok hozzárendelése a csoporthoz

Bár semmi akadálya annak, hogy egyedül használd a létrehozott projektedet – bármilyen meglepő, még egyetlen fejlesztő esetében is komoly előnyökkel járhat ez –, egy fejlesztőcsapat felépítéséhez további felhasználói fiókokat kell rendelned a projekthez.

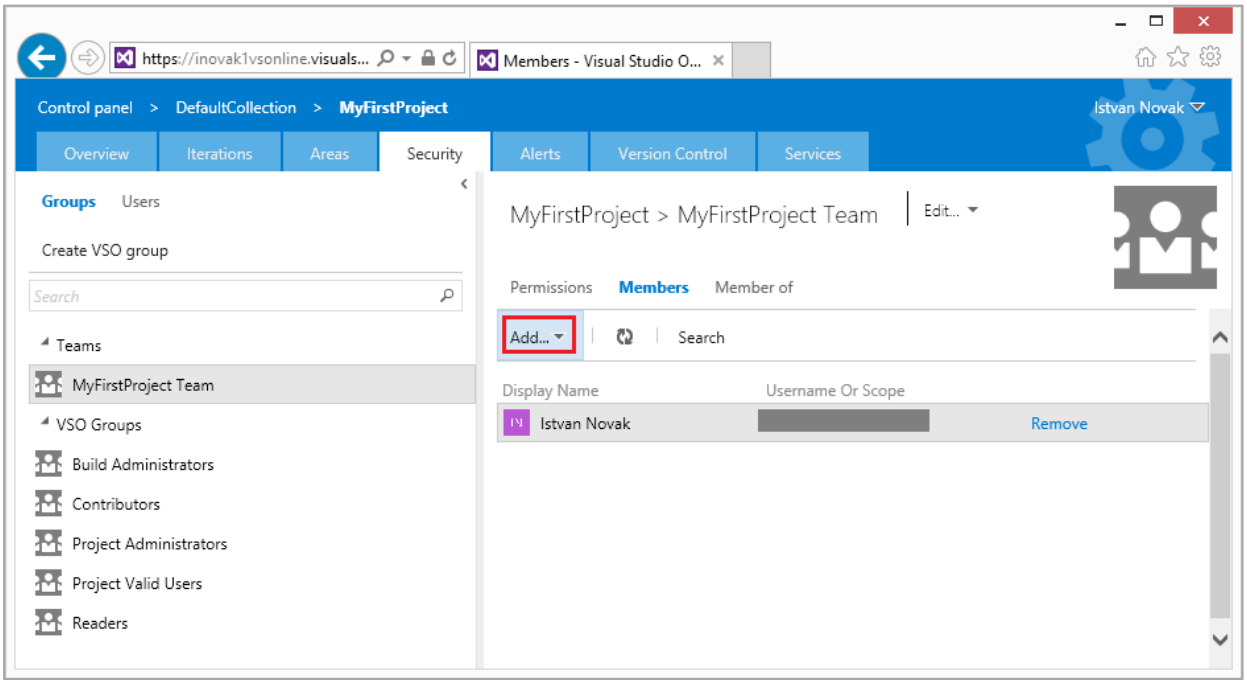
Az alábbi lépéseket követve ezt igazán egyszerűen megteheted:

1. A Visual Studióban a Team Explorer ablak fejrészében kattints a *Home* (kis ház alakú) ikonra, majd ezután a *Settings* gombra! A Team projekt listában lévő funkciók közül válaszd a *Group Membership* linket (3-13 ábra)! Ez a portál csoporttagságokat adminisztráló részére irányít.



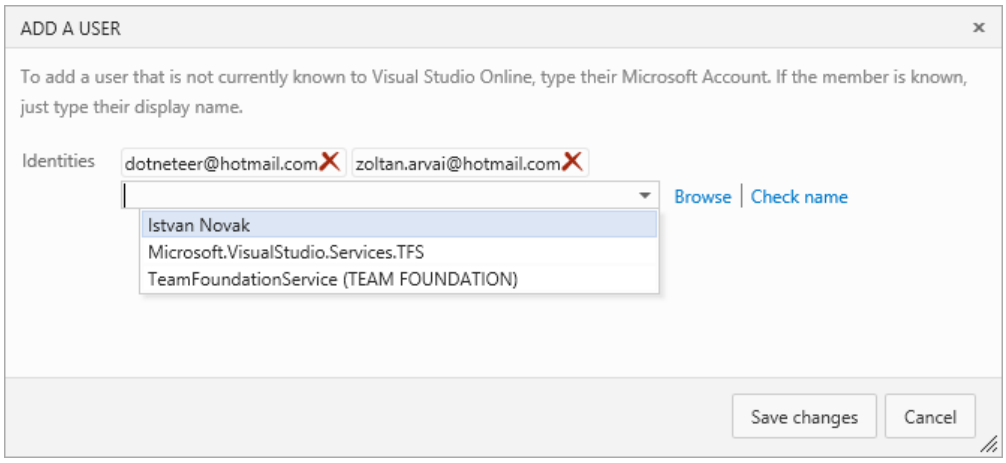
3-13 ábra: A csoporttagságok beállításának elérése

2. A portálon megjelenik a projekt biztonsági beállításait tartalmazó lap, amint azt a 3-14 ábra mutatja. Ez két panelből áll: a bal oldalon a csoportokat (alapértelmezetten ez jelenik meg) vagy a felhasználókat tartalmazó lista, a jobb oldalon pedig a bal oldalon kiválasztott elem tulajdonságai jelennek meg. A jelen esetben a *MyFirstProject Team* látható, illetve annak felhasználói.



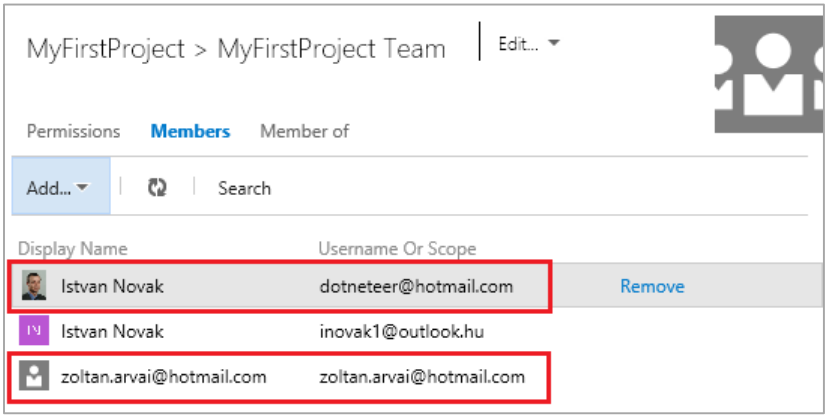
3-14 ábra: A biztonsági beállítsok oldala a projektportálon

3. Kattints a képernyőn is megjelölt *Add* gombra, majd a megjelenő lenyíló listán az *Add user* funkcióra! A megjelenő dialógusban add meg a projekthez rendelt felhasználókat (3-15 ábra)! Ha egy felhasználó még nem tagja a projektnek (a felhasználó a projekt más csoportjához is tartozhat), egyszerűen gépeld be Microsoft Account azonosítóját, majd kattints a *Check name* linkre! Ha a felhasználó már tagja a projektnek, akkor a lenyíló listából választhatod ki, illetve a *Browse* link segítségével keresheted ki.



3-15 ábra: Felhasználók hozzárendelése a projekthez

4. Ha az új felhasználókat felsoroltad, kattints a *Save changes* gombra! A csoport új tagjai ezután megjelennek a listában, ahogyan azt a 3-16 ábrán láthatod.



3-16 ábra: Új felhasználók a tagok között

Ezzel készen is vagy. Az így hozzáadott felhasználók teljes jogú tagjai a projektnek, vagyis minden tevékenységet – az adminisztratív funkciók kivételével – korlátlanul elvégezhetnek.

Jogosultságok kezelése

Természetesen egy csapatban nem szükségszerűen bír minden felhasználó teljes jogosultságokkal. Ha a fent leírtnál finomabb jogosultságok kezelésére van szükséged, a projektportálon azt is megteheted. Sokfajta módon alakíthatod ki a saját jogosultsági rendszeredet, és az ehhez szükséges műveletek egyszerűen elérhetők a portálon keresztül – illetve a Visual Studio környezetéből a Team Exploreren keresztül is kezdeményezhetők. Ahhoz, hogy ezt megtehesd, érdemes néhány fontos dolgot – a teljesség igénye nélkül – megtanulnod a Visual Studio Online jogosultságkezelési rendszeréről.

A projektedhez tartozó csapat (a 3-14 ábrán ez a *MyFirstProject Team*) felhasználókból és Visual Studio Online csoportok (*VSO groups*) tagságából épül fel. A VSO csoportok tagjai szintén egyéni felhasználók vagy más csoportok lehetnek.

A csoportokhoz és a felhasználókhoz számtalan elemi jogosultság tartozhat, ezek eredője határozza meg azt, hogy egy adott felhasználó milyen lehetőségekkel rendelkezik egy adott projekten. Minden jogosultság az alábbi értékek valamelyikével rendelkezhet:

- **Nincs beállítva (*Not Set*)**. A jogosultság az adott felhasználóra (csoportra) nincs beállítva (sem tiltva, sem engedélyezve). A felhasználó más csoporttagságokon keresztül örökölheti a beállítás értékét.

- **Tiltott (*Deny*)**. Az adott csoport vagy felhasználó nem használhatja az érintett műveletet – még akkor sem, ha más csoportokon keresztül arra engedélye volna. Ez azt jelenti, hogy a tiltás mindenfajta engedélyezésnél erősebb.
- **Engedélyezett (*Allow*)**. Az adott csoport vagy felhasználó elérheti az érintett műveletet.

A csoportokon keresztül az engedélyezett műveletek öröklődhetnek. Ez azt jelenti, hogy ha egy felhasználó adott jogosultsága nincs beállítva, de az egyik szülőcsoportjában (vagyis olyan csoportban, amelynek a felhasználó közvetve vagy közvetlenül a tagja) engedélyezett, akkor a felhasználó elvégezheti az adott műveletet.

Minden projekt létrehozásakor automatikusan létrejönnek a 3-1 táblázatban leírt projektcsoportok.

3-1 táblázat: VSO szintű csoportok

Csoport	Leírás
Build Administrators	A csoport tagjai létrehozhatnak, módosíthatnak és törölhetnek kódépítési (*build*) definíciókat, elindíthatják és kezelhetik azokat.
Contributors	A csoport tagjai létrehozhatnak, módosíthatnak és törölhetnek a projekthez tartozó bejegyzéseket.
Project Administrators	Ennek a csoportnak a tagjai minden, a projekthez kapcsolódó műveletet elvégezhetnek. A csoporthoz tartozó jogosultságokat nem lehet megváltoztatni.
Project Valid Users	A csoport tagjai hozzáférhetnek a projekthez.
Readers	A csoport tagjai hozzáférhetnek a projekthez, de alapértelmezett módon csak a bejegyzések megtekintésére van jogosultságuk, a módosításra nem.

A portálon minden felhasználóhoz és csoporthoz megtekinthető az effektív jogosultságok listája, illetve az, hogy az adott felhasználó vagy csoport mely csoportoknak tagja. A csoportok esetén ezenkívül még tagsági listájuk is megjeleníthető.

> *A jogosultságokkal kapcsolatos további részletekről az alábbi weblapon tájékozódhatsz:*
> ***`http://msdn.microsoft.com/en-us/library/ms252587.aspx`***.

A Visual Studio Online képességei

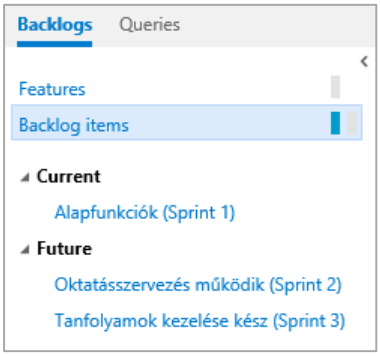
A könyv további fejezeteiben a Visual Studio Online összes – a csapatmunka szempontjából kiemelkedően – fontos képességével találkozhatsz, azok használatát is megismerheted. A fejezetnek ebben a részében egy rövid áttekintést kapsz arról, hogy milyen konkrét feladatokra is használhatod ezt a terméket.

Termékek, sprintek, backlog

Amint azt a második fejezetben már megtanultad, a Scrum alapját a csapat előtt álló feladatok jelentik. A Visual Studio Online lehetővé teszi, hogy a csapat előtt álló feladatokat több szinten kezelhesd. A fejlesztés alatt álló terméked kapcsán kezelheted

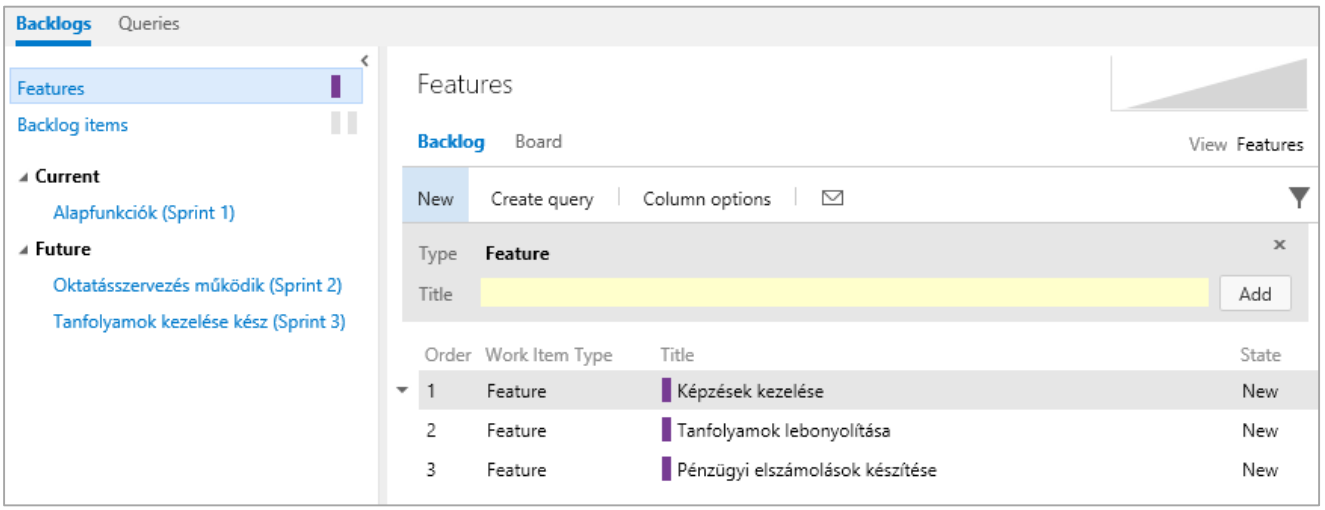
- a termék egészét átfogó eposz jellegű backlog bejegyzéseket, amelyek további lebontásra szorulnak;
- a termékhez tartozó, a csapat által önmagában kezelhető méretűnek ítélt backlog bejegyzéseket;
- a termékkibocsátáshoz kötődő változatokat, illetve az azok előállításához kapcsolódó sprinteket.

A 3-17 ábrán a változatok és sprintek kezelésére láthatsz egy példát. A termékfejlesztésnek ez a ciklusa három jól megkülönböztethető sprintből áll. A termékhez tartozó eposzokat a *Features* listán, a már lebontott backlog bejegyzéseket a *Backlog items* listán lehet megtekinteni.



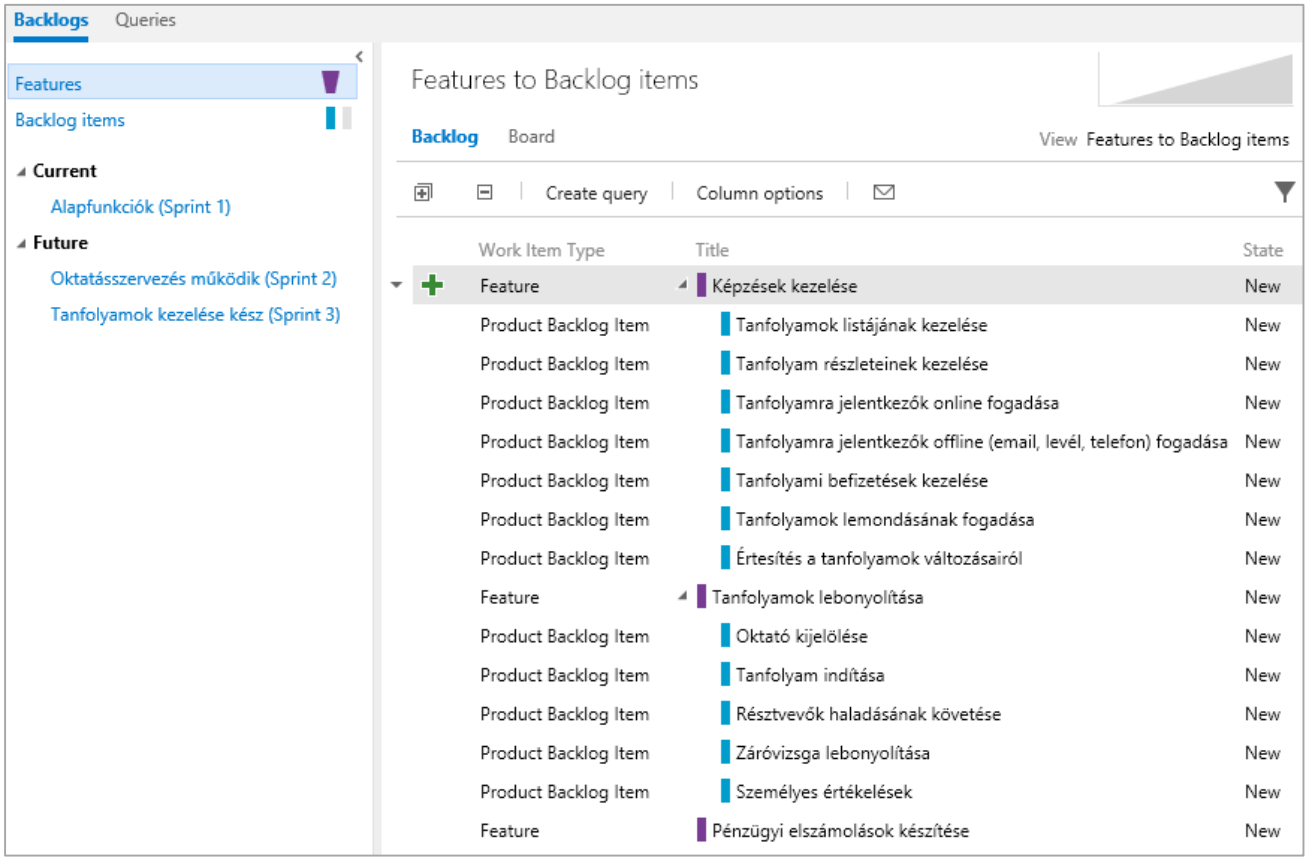
3-17 ábra: Sprintek kezelése

A fejlesztő csapattal együtt dolgozó p*roduct owner* a 3-18 ábrán látható képességeket definiálta a fejlesztési projekt előkészítése során.



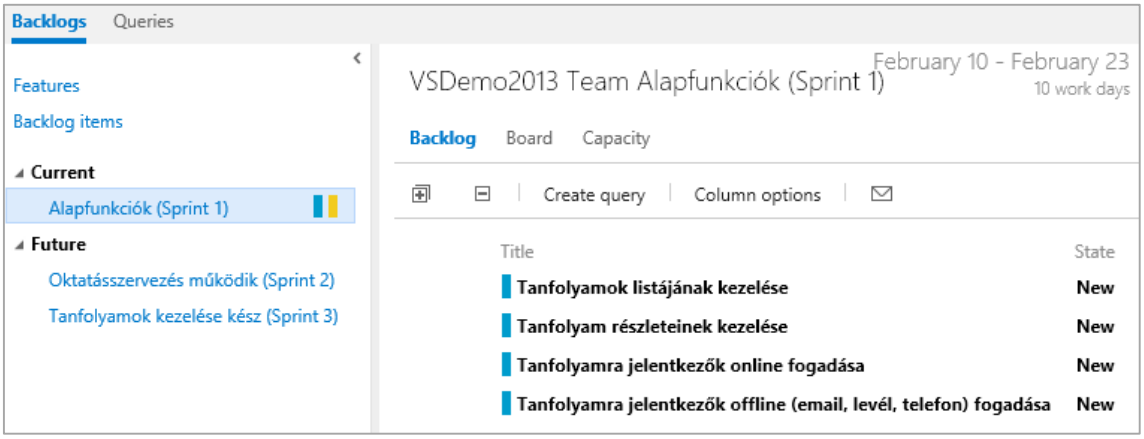
3-18 ábra: A termékhez tartozó eposzok

A csapat ezekből az eposzokból az első kettőt már önmagukban kezelhető backlog elemekre bontotta le, amint azt a 3-19 ábra mutatja. A harmadik eposzt (Pénzügyi elszámolások készítése) a csapat még nem bontotta le, ezt majd a *backlog grooming* tevékenységek során teszik meg, hogy mire a fejlesztés megfelelő szakaszához érnek, itt is jól definiált feladataik legyenek.



3-19 ábra: A lebontott eposzok

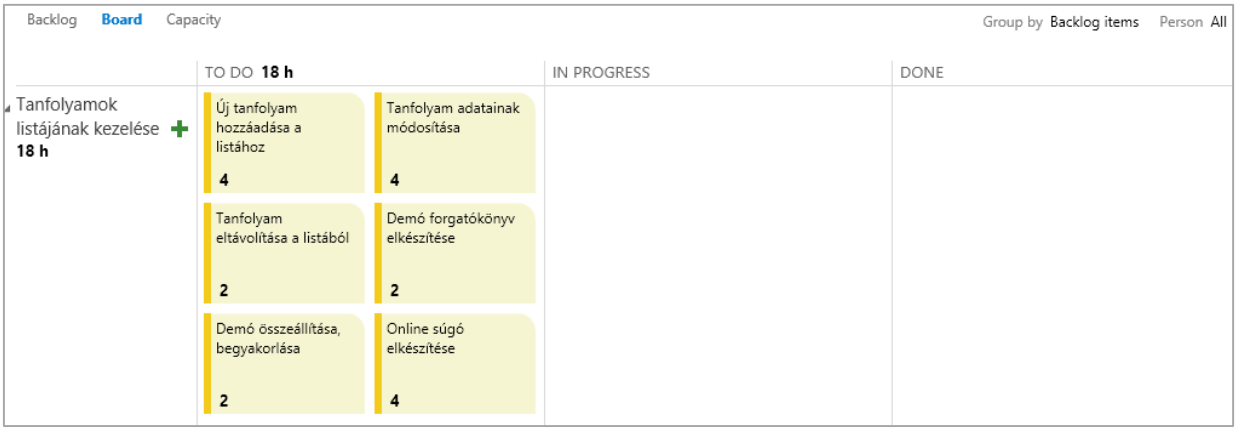
Az első sprint előkészítése során a csapat a 3-20 ábrán látható backlog elemeket sorolta be az elvégzendő – és a csapat által felvállalt – tevékenységek közé.



3-20 ábra: A csapat ezeket a backlog elemeket vállalja az első sprint során

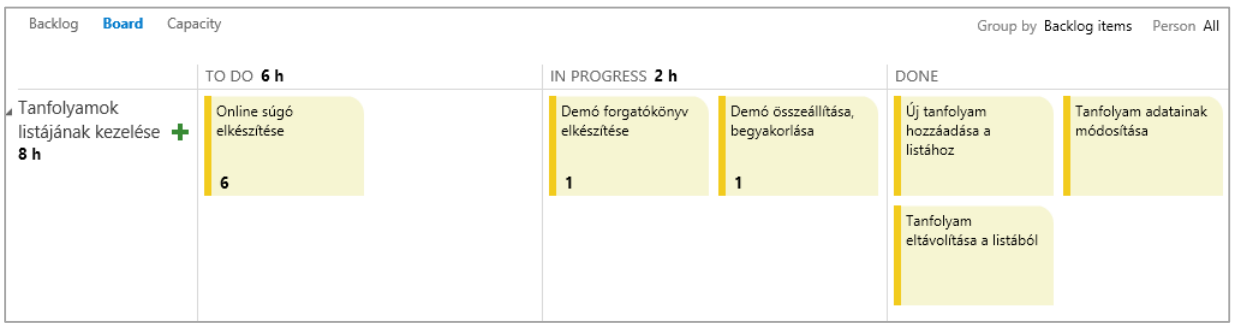
Feladatok kezelése

Amint azt megtanultad, a csapat arra törekszik, hogy a sprint során a felvállalt backlog elemek mindegyikét sikeresen, az átvételi kritériumoknak és „kész" fogalmának (*definition of done)* meghatározásában foglaltaknak megfelelően végezze. A sprint indítása előtt a csapat a backlog elemeket konkrét feladatokra bontja le. A 3-21 ábrán egyetlen backlog elem feladatait láthatjuk, mindegyik tartalmazza a teljes végrehajtás – a csapat által közösen megbecsült és elfogadott – időtartamát órákban.



3-21 ábra: A backlog elemek lebontása feladatokra

A sprint indítása után a csapat folyamatosan követi az egyes feladatok előrehaladását, amint azt a 3-22 ábra is mutatja. Ez a pillanatfelvétel azt az állapotot örökíti meg, amikor a csapat az alapvető funkciókat már megvalósította, és éppen a demóra készül, illetve a feladat teljesítéséhez szükséges online súgó elkészítésére.



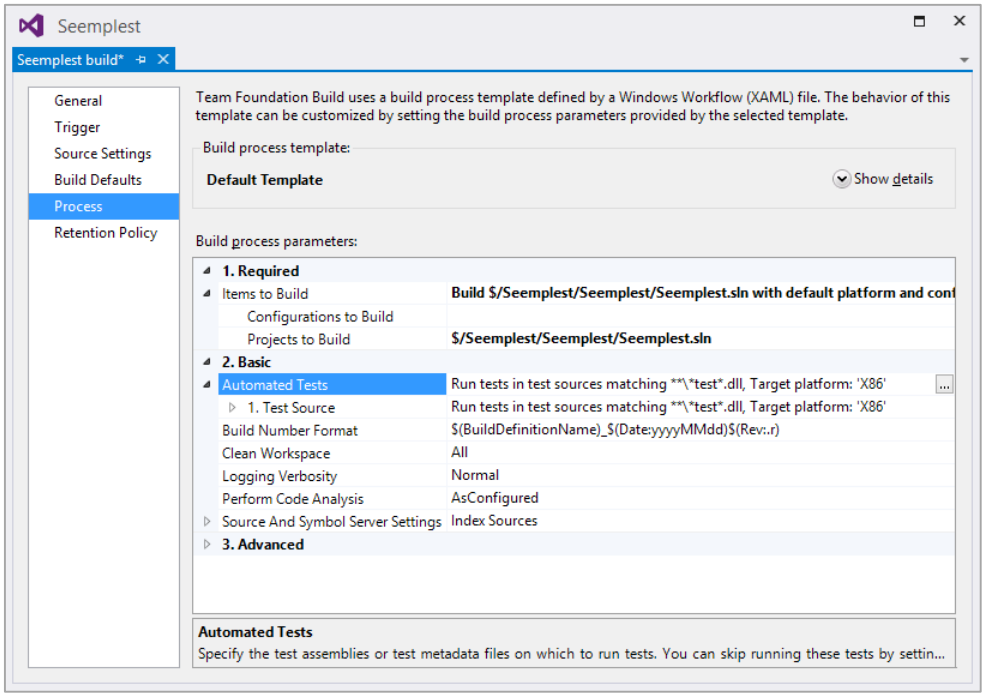
3-22 ábra: A feladatok állapota a sprint közben

Az ábrán jól látható, hogy a csapat a demó forgatókönyv elkészítését és a demó begyakorlását végzi éppen. Azt is észre lehet venni, hogy bár még hozzá sem kezdtek az online súgó elkészítéséhez, de felismerték, hogy az eredetileg tervezett 4 óra erre nem lesz elég, és a becsült ráfordítást 6 órára módosították.

Forráskódok verzióinak kezelése, automatikus fordítás

Szoftverfejlesztő környezetről lévén szó, a Visual Studio Online támogatja a forráskódok kapcsán az összes fontos verziókezelési funkciót egészen a kódok frissítésétől (*check-in*, *check-out*) a kódágak kezelésén át (*branching*, *merging*) az eseti változatok támogatásáig (*shelving*, *unshelving*). Ezek a funkciók a Visual Studio környezetéből éppen úgy elérhetők, mint a projektportálról.

Az agilis módszerek – így a Scrum – támogatásának sokkal fontosabb eleme, hogy a Visual Studio Online lehetővé teszi a kódépítési folyamat (*build definition*) létrehozását, testreszabását. Ennek segítségével a termék forráskódjának minden a verziótárba visszaírt változtatásakor a szerver oldalon lefut egy automatikus kódépítési eljárás, amely a forrás fordítása mellett további minőségbiztosítási tevékenységeket, például az automatikus tesztek futtatását, a kód statikus elemzését is elvégezheti. A 3-23 ábra éppen egy ilyen folyamat létrehozását mutatja be.



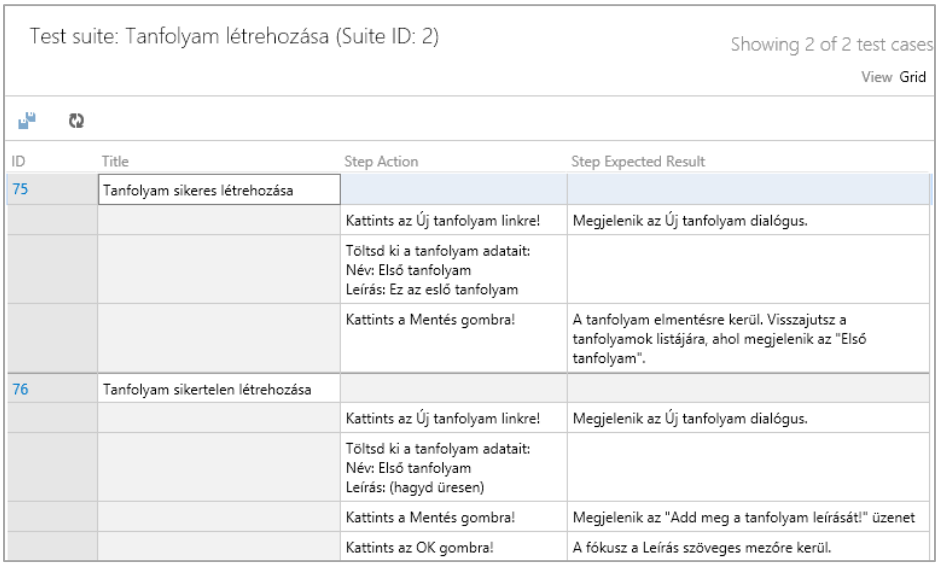
3-23 ábra: Egy kódépítési folyamat definiálása

A projekthez az alapértelmezett sablon mellett még több is érkezik a Visual Studio Online-nal, van például olyan sablon, amely az Azure alkalmazások automatikus ellenőrzését és – a minőségi kritériumok teljesítése után – azok telepítését is elvégzi.

A kódépítési folyamat egyik erőssége, hogy lehetővé teszi az ún. *continuous deployment* szemlélet használatát, vagyis azt, hogy a termékfejlesztés során biztosítani tudjuk, hogy rendszeresen (természetesen a megfelelő minőségbiztosítás után), akár naponta módosíthassuk az éles környezetben lévő alkalmazást. A minőségvezérelt szemlélet egészen odáig fokozható, hogy a minőségi kritériumoknak meg nem felelő kód – vagyis az, amely „megtöri” a korábbi sikeres kódépítési és ellenőrzési folyamatot – vissza sem kerülhet a verziótárba.

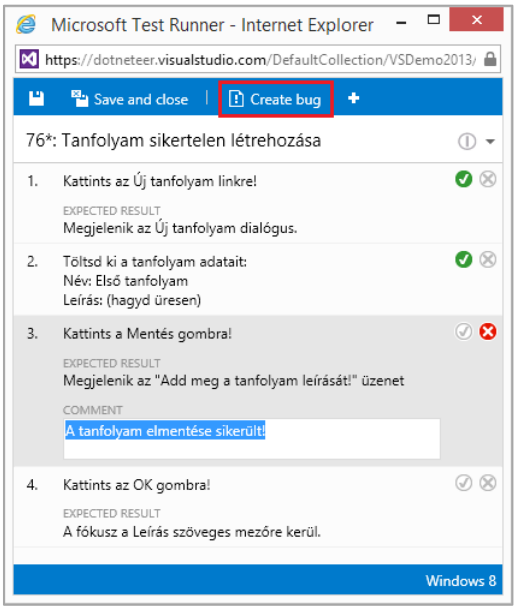
Tesztelés

A kódépítéshez tartozó automatikus tesztelés mellett a Visual Studio Online támogatja a manuális tesztek tervezését és futtatását egyaránt. A 3-24 ábra a manuális tesztek tervezését demonstrálja.



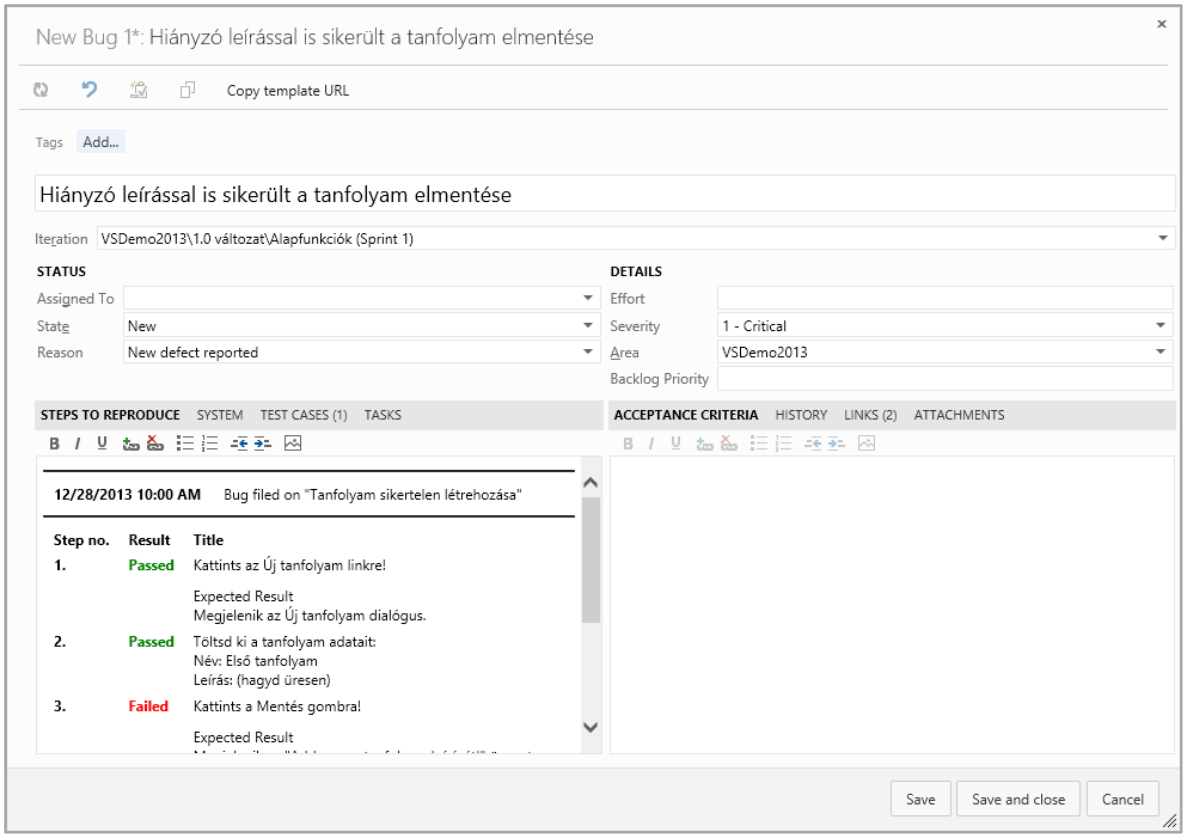
3-24 ábra: Tesztesetek tervezése

Az elkészített eseteket a csapat tesztelést végző tagja – ez természetesen az adott funkció fejlesztője is lehet – közvetlenül a projektportálról vagy akár a Microsoft Test Manager eszközéből is futtathatja. A 3-25 ábrán a „Tanfolyam sikertelen létrehozása” tesztesetet látjuk végrehajtás közben. A tesztelő éppen a harmadik lépés ellenőrzésénél jár, amikor hibát tapasztal.



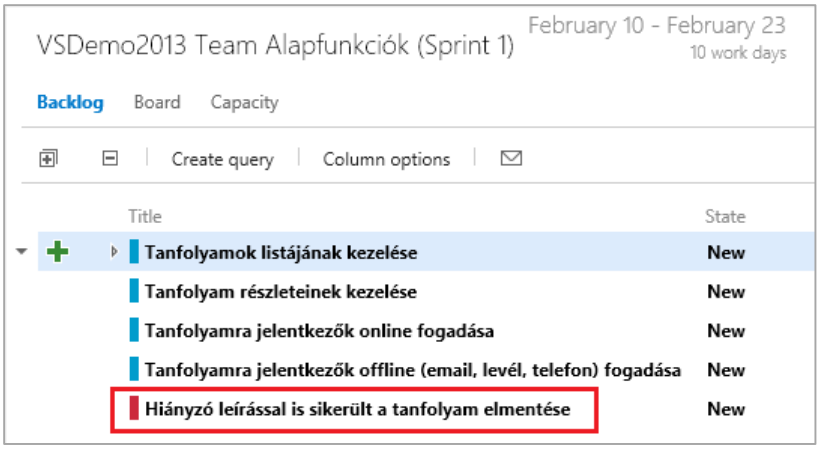
3-25 ábra: A teszteset végrehajtása

Ezt a hibát azonnal jelezheti a *Create bug* linkre kattintással. A portálon megnyílik az a dialógus, amellyel a hiba adatai megadhatók (3-26 ábra). Az ábrán jól látható, hogy a probléma előállításához szükséges részleteket a tesztelést végző eszköz automatikusan bemásolja a bejegyzésbe.



3-26 ábra: Hibajelentés létrehozása

A Visual Studio Online lehetővé teszi, hogy a tesztesetek összeköthetők legyenek azokkal a backlog bejegyzésekkel, illetve feladatokkal, amelyekhez tartoznak. Értelemszerűen a hibák automatikusan összekötésre kerülnek azokkal a tesztesetekkel, amelyek során jelentkeznek. Ha a csapat úgy dönt, hogy a hiba kijavítása nélkül a sprint nem tekinthető teljesnek, akkor azt a sprinthez rendelheti, amint azt a 3-27 ábra is mutatja.



3-27 ábra: A hiba a sprint backlogjába kerül

Kibocsátási ciklusok kezelése

A termékekhez tartozó kibocsátási ciklusok kezelésére a Microsoft egy önálló terméket, a Release Managert biztosítja, amely beleintegrálódik a Visual Studio folyamataiba. A termék három komponensből áll:

- A kibocsátási folyamatok kezelését biztosító szerver oldali funkciók
- A folyamatok definiálását, vezérlését és kezelését biztosító kliens oldali alkalmazás
- Deployment agent: a telepítés célját jelentő számítógépeken futó ügynökalkalmazás

Ez a termék hasznos eszköz lehet a continuous deployment megvalósítására azoknak a csapatoknak a kezében, akik már járatosak az agilis módszertanok használatában.

Alkalmazások monitorozása

A Visual Studio Online egy új (még csak előzetes állapotban) elérhető szolgáltatása az Application Insights, amely lehetővé teszi, hogy az alkalmazást éles környezetében figyeld meg, mérve annak teljesítményjellemzőit. Akár felhőben, akár a saját adatközpontodban futtatod az alkalmazást, már a fejlesztés első fázisaitól kezdve egészen az éles üzemben folyamatosan nyomon követheted azt, és így biztos lehetsz abban, hogy az alkalmazás működése megfelel a teljesítményhez kapcsolódó követelményeknek.

Jelenleg az Application Insights még csak külön meghívókóddal érhető el, amelyre egy-két hetet kell várni.

A Visual Studio 2013 újdonságai – egyéni produktivitás

Az agilis fejlesztés hatékonyságát nemcsak a csoportmunka támogatásán keresztül lehet növelni, hanem olyan eszközökkel is, amelyek a fejlesztők egyéni produktivitását javítják. A Visual Studio 2013-ba több ilyen képesség is bekerült, ezek közül mutatunk be itt néhányat – a teljesség igénye nélkül.

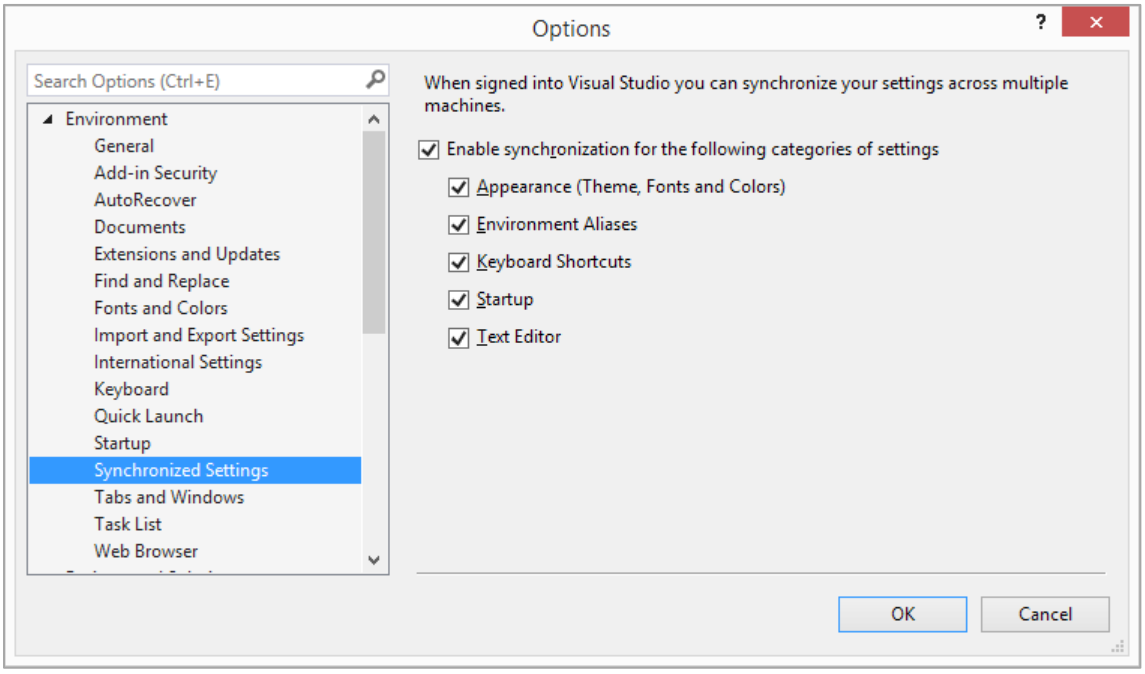
Felhasználói fiókok kezelése

Amint azt a fejezet elején már bemutattuk, a Visual Studio indításakor Microsoft Account azonosítód használatával bejelentkezhetsz az alkalmazásba. Ez a bejelentkezés több lehetőséget is ad a kezedbe:

- Könnyedén – újabb azonosítás nélkül – hozzáférhetsz azokhoz a Visual Studio Online erőforrásokhoz, amelyekre a felhasználói fiókodnak jogosultsága van.
- A fejlesztői környezet beállításai több gép között is szinkronizálásra kerülnek.

Ma már a legtöbb fejlesztő több számítógépet is használ a teljes fejlesztési folyamat során. Az egyéni hatékonyság javítása érdekében minden fejlesztő igyekszik saját környezetét úgy kialakítani (ablakok elrendezése, színvilág, kódszerkesztő beállításai stb.), hogy az a lehető legjobban kézre álljon.

Korábban ezeket a beállításokat kézzel kellett szinkronizálni (export, majd import), most azonban ez már teljesen automatikusan történik. A *Tools → Options* dialógus *Environment → Synchronized Settings* kategóriájában – amint azt a 3-28 ábra mutatja – beállíthatod, hogy ez a szinkronizálás a fejlesztőkörnyezet mely beállításaira terjedjen ki.



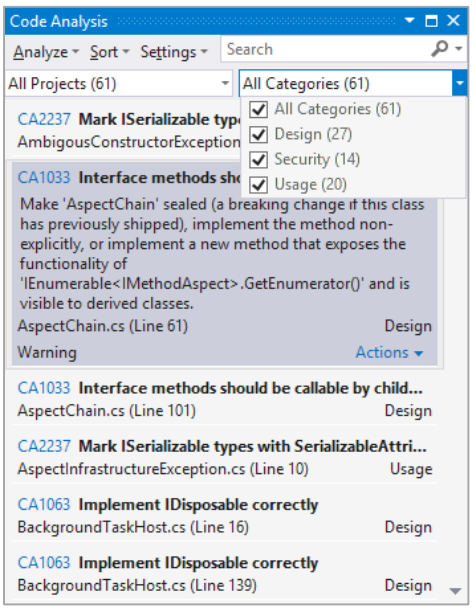
3-28 ábra: A szinkronizálás beállításai

A kódminőség javítása

Az összes agilis módszertan komoly hangsúlyt helyez a kód minőségének javítására, mint a hatékony fejlesztés egyik sarokkövére. Ez azt jelenti, hogy a fejlesztők által elkészített kódnak nemcsak a funkcionális követelményeket kell kielégítenie, hanem támogatnia kell a kód tesztelhetőségét, karbantarthatóságát. Ez biztosítja azt, hogy a fejlesztőcsapat mindig ismert – és az előre tervezett feladatok szempontjából jól kezelhető – kódminőséggel haladhasson tovább.

A kód minősége alapvetően szubjektív dolog, de természetesen sok olyan objektív mérési és ellenőrzési technika van, amelyek mind a szubjektív minőség javításának irányában hatnak. A Visual Studio ezek közül – az automatikus tesztelés teljes körű támogatása mellett – többet is elérhetővé tesz. Ezekből itt három olyat szeretnénk kiemelni, amelyek egy agilis módszerekkel dolgozó csapat számára nélkülözhetetlenek: a kódanalízist, a kódmetrikák felhasználását és az ismétlődő kódrészek kezelését.

A kódanalízis szerepe az, hogy segít felismerni azokat a potenciális problémákat (figyelmetlenségek, elkerülendő kódminták és programozási gyakorlatok), amelyek később nehezen felderíthető stabilitási vagy karbantarthatósági kérdéseket vethetnek fel. Az *Analyze → Run Code Analysis on Solution* parancs segítségével (Alt+F11) elindíthatod ezt az elemzést. Egy ilyen művelet eredménye látható a 3-29 ábrán.



3-29 ábra: Kódanalízis eredménye

Az ábrán a sötétített háttérrel jelölt észrevétel csak egy abból a 61-ből, amelyet az elemzés feltárt. Az ábrán látható, hogy az elemzés ezeket kategóriákba sorolta, 27 tervezési, 14 biztonsági és 20 felhasználási problémára hívja fel a fejlesztő figyelmét. Minden potenciális probléma kódjára kattintva arra a weblapra jutunk, amely ismerteti a problémát, annak jelentőségét és kiküszöbölésének módját. A problémára kattintva azonnal az érintett kódrészhez navigálhatunk.

A bejegyzés jobb alsó sarkában található *Actions* lenyíló lista segítségével a talált probléma további sorsáról dönthetünk. Ha a probléma nem valós – tudjuk, hogy az adott környezetben annak nincs jelentősége –, elnyomhatjuk annak további megjelenítését, és így a későbbi elemzések már nem fogják azt kiemelni. Lehetőségünk van arra is, hogy a probléma kapcsán feladatot (backlog bejegyzést, bug jelentést stb.) hozzunk létre, és azt hozzárendeljük egy sprinthez és/vagy egy csapattaghoz.

A kód minőségét, karbantarthatóságát mérőszámokkal, ún. kódmetrikákkal is jellemezhetjük. Ezek számítását az *Analyze → Calculate Code Metrics for Solution* paranccsal indíthatjuk el. A 3-30 ábra egy projekt kódmetrikáit mutatja be. A kód egyes részeihez tartozó kódmetrikákat egészen a kódban definiált típusokhoz tartozó egyedi műveletekig lebonthatjuk.

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
One or more projects were skipped. Code metric					
Core\Seemplest.Core (Debug)	91	1 983	4	382	3 088
MsSql\Seemplest.MsSql (Debug)	80	589	5	143	1 290
Seemplest.MsSql.Configuration	93	3	5	4	4
Seemplest.MsSql.DataAccess	80	465	1	117	1 030
Seemplest.MsSql.Queue	78	93	2	26	225
SqlNamedQueue	66	45	1	21	121
SqlNamedQueueProvider	72	31	1	16	74
SqlPoppedMessage	81	5	2	6	13
SqlQueuedMessage	91	12	1	4	17
Seemplest.MsSql.Records	90	25	2	8	25
Seemplest.MsSql.Tracing	73	3	2	6	6
Samples\BackgroundTaskHostSample (Debug)	80	36	4	34	76
UnitTests\Seemplest.Core.UnitTests (Debug)	89	898	5	331	3 226
UnitTests\Seemplest.MsSql.UnitTests (Debug)	86	688	2	165	2 587

3-30 ábra: Egy projekt kódmetrikái

Az egyes mérőszámok jelentését a 3-2 táblázat foglalja össze.

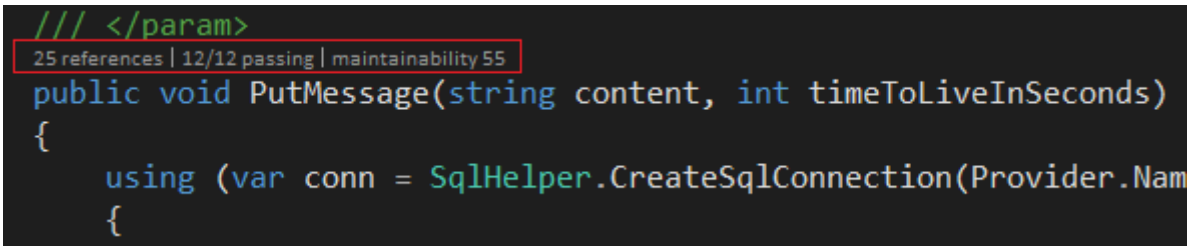
3-2 táblázat: A Visual Studio által számított kódmetrikák

Metrika	Jelentés
Maintainability Index	*Karbantarthatósági index*. Olyan indexértéket számít ki (0 és 100 között), amely a kód karbantarthatóságának relatív egyszerűségét fejezi ki. Minél nagyobb ez az érték, annál jobban karbantartható a kód. A vörös színnel jelölt kódrészek (0-9) azonnal beazonosítható problémás pontokra utalnak a kódban. A sárgával jelölt részek (10-19) azt mutatják, hogy az adott kód mérsékelten karbantartható. A zöld szín (20-100) arra utal, hogy a megjelölt kód könnyen karbantartható.
Cyclomatic Complexity	*Ciklometrikus komplexitás*. A kód szerkezetének bonyolultságát méri, a program lehetséges végrehajtási útjainak a számát adja meg az adott kódfolyamban. Minél nagyobb ez a szám, annál összetettebb a program és így annál több tesztelésre van szükség a jó kódfedettség eléréséhez, illetve annál kevésbé karbantartható.
Depth of Inheritance	*Öröklődési mélység*. Azoknak az osztálydefinícióknak a számát adja meg, amelyek ahhoz szükségesek, hogy a típusdeklarációtól a hierarchia gyökeréig eljussunk. Minél mélyebb ez a hierarchia, annál nehezebb lehet megérteni, hogy adott műveletek, mezők és tulajdonságok hol vannak definiálva, illetve újradefiniálva.
Class Coupling	*Osztálycsatolás*. Az egyedi osztályok csatolását méri paramétereken, lokális változókon, visszatérési értékeken, metódushívásokon, generikus típusok példányosításán, ősosztályokon, interfészek megvalósításán, mezők és külső típusok definiálásán, valamint attribútumok használatán keresztül. A jó szoftvertervezési elvek azt diktálják, hogy a típusok és műveleteik között magas kohéziónak és kevés csatolásnak kell lennie. A műveletekhez kapcsolódó magas csatolási szám azt jelzi, hogy nehezen újrahasznosítható és karbantartható kóddal (tervezéssel) állunk szemben, mert túl sok a más típusoktól való függőség.
Lines of Code	*Kódsorok száma*. Ez a metrika a kódsorok megközelítő számát mutatja. Ez a szám nem a forráskód sorainak valódi számán alapszik (ez függhet a kódolási stílustól), hanem a generált közbenső kódból (IL kód) áll elő. A nagyon magas szám azt jelzi, hogy egy típus vagy művelet túl sok dolgot próbál csinálni, és inkább kisebb darabokra kellene azt szétbontani.

Az ismétlődő kódrészek felismerése hasznos funkció, hiszen ez az adott részek refaktorizálására ad lehetőséget. Az *Analyze → Analyze Solution for Code Clones* funkció segítségével megtalálhatjuk ezeket a programrészleteket, és kiemelhetjük, átalakíthatjuk azokat.

A Code Lens használata

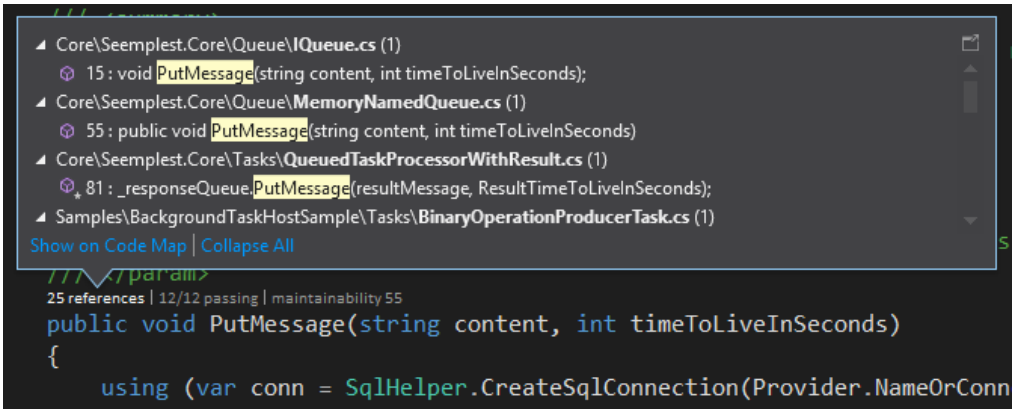
A Visual Studio 2013 Ultimate változatában megjelent Code Lens funkció komoly segítséget jelenthet a fejlesztők számára a kód használata, szerkesztése, megértése során. Ez a funkció a típusok, műveleteik, tulajdonságaik és egyéb tagjaik kapcsán fontos információkat jelenít meg, ahogyan azt a 3-31 ábra is mutatja.



3-31 ábra: Code Lens információ megjelenítése

A Code Lens információ a típusok és műveletek fölött szerepel, ahogyan azt a 3-31 ábrán a bekeretezett rész is mutatja. Ebből például azonnal látható, hogy a **`PutMessage`** műveletre 25 helyen van hivatkozás a forráskódban, és mind a 12 teszteset, amely ezt a műveletet használja, sikeresen lefutott. Azt is megállapíthatjuk, hogy a művelet karbantarthatósági indexe 55 (a 0-100 skálán).

A megjelenített információk linkként is működnek. A 3-32 ábra azt mutatja, hogy a hivatkozások számára kattintva meg is tekinthetjük azokat. Ezek a hivatkozások természetesen nemcsak információt nyújtanak, hanem egyúttal segítik is fejlesztő munkáját: elegendő egy dupla kattintás az adott hivatkozásra, és a kódszerkesztő máris az adott helyre navigál.



3-32 ábra: Kódhivatkozások megtekintése a Code Lens segítségével

A Code Lens beépített funkcionalitása kiterjeszthető. A fejlesztőknek lehetőségük van olyan kiegészítések írására, amelyek beilleszkednek a Code Lens infrastruktúrájába és további információt jelenítenek meg az adott típusról, illetve műveletről, valamint további funkciókat tesznek elérhetővé. A fenti ábrákon a kód karbantarthatóságára vonatkozó információ ilyen – a *Tools → Extensions and Updates* funkcióval letölthető – kiegészítés révén valósul meg.

Összegzés

A Visual Studio Online rengeteg olyan hasznos funkciót tartalmaz, amely az agilis fejlesztést – és az ahhoz kapcsolódó csoportmunkát – segíti. Kisméretű (legfeljebb 5 tagú) csapatok számára ingyen elérhető a Visual Studio Express eszközeivel. A csapattagok a Microsoft Account fiókjuk segítségével használhatják a szolgáltatást, amely a felhőben tárolja a csapatmunka alapját jelentő listákat, bejegyzéseket, illetve a forráskód verzióit.

A Visual Studio Online beépített módon támogatja a Scrum legújabb változatának használatát segítő folyamatsablont, ezt egy új projekt létrehozása során tudjuk kiválasztani.

A csapat tagjai a Visual Studio környezetében felhasználói fiókjukkal be tudnak jelentkezni az online szolgáltatás környezetébe, és a csapatmunkához használt funkciókat nemcsak a portálon, hanem a fejlesztőkörnyezeten keresztül is elérhetik.

A következő fejezetben megismerkedhetsz a Scrum használatának alapvető eszközével, a backloggal, és megtanulhatod, hogyan használható az a kifejlesztendő termék képességeinek kezelésére.

B. Melléklet: A munkadarabok kezelésének testreszabása

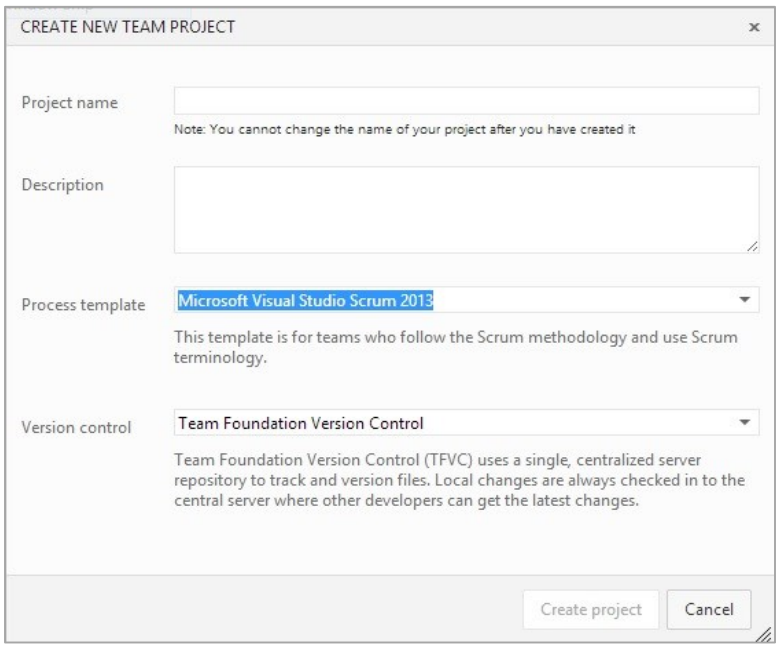
Ebben a mellékletben a Team Foundation Server munkadarab követési (*Work Item Tracking*) lehetőségeit ismerheted meg. Tárgyaljuk a munkadarabok felépítését, a mögöttük rejlő folyamatokat és modelleket, illetve ennek a testreszabási lehetőségeit:

- A folyamatsablon és a munkadarabok kapcsolata
- A munkafolyamat testreszabása
- A munkadarabok testreszabásának folyamata
- Mezők, felhasználói felület, kapcsolódó állapotgépek

A Team Foundation Server óriási segítséget nyújt a csapatmunka koordinálásában és követésében. A termék által biztosított egyik legfontosabb eszköz a munkadarab (*work item*). A munkadarabok (task, bug stb.) módszertanonként eltérőek lehetnek, de pontos vezetésük és aktív használatuk nagymértékben hozzájárul a projekt követhetőségéhez, valamint a csapatmunka hatékonyságának növeléséhez.

A folyamatsablon és a munkadarabok kapcsolata

Amikor a fejlesztés, illetve a megelőző tervezési munkák elkezdődnek, a projekthez létrehozunk egy *team project*et. Ez egymagában összefogja a projekthez kapcsolódó összes értéket, például a forráskódot, a dokumentációt, az iterációk terveit, a termék backlogot és a munkadarabokat. A fentiekből az következik, hogy ennek a projektnek követnie kell azt a módszertant, amit a csapat előre kiválaszt, annak érdekében, hogy a munkafolyamatokat koordinálni tudja. A projekt létrehozásakor ún. folyamatsablont (*process template*) is választunk (B-1 ábra).

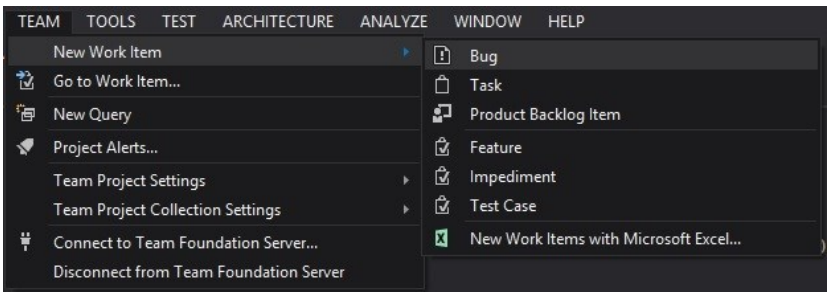


B- 1 ábra: Folyamatsablon kiválasztása a projekt létrehozásának pillanatában

A folyamatsablon definiálja a kiválasztott munkamódszernek, illetve a módszer fogalomrendszerének megfelelő dokumentumokat, logikai egységeket. Ez azt jelenti, hogy ha a Scrum sablont választjuk, akkor más típusú dokumentumokat, riportokat, munkadarab típusokat fogunk kapni, mint ha mondjuk a CMMI-t megvalósító sablont választottuk volna.

A folyamatsablon tartalmazza, hogy:

- milyen munkadarab típusok léteznek (B-2 ábra),
- azok milyen attribútumokkal rendelkeznek,
- az egyes attribútumok milyen értékeket vehetnek fel,
- az egyes értékek miképp követhetik egymást, vagyis definiálja, hogy milyen folyamat és állapotátmenetek lehetségesek az egyes attribútumok között.



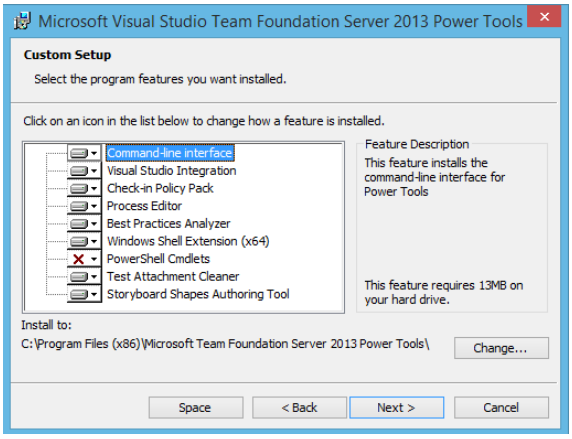
B- 2 ábra: Scrum folyamatsablon által definiált munkadarab típusok

A munkafolyamat testreszabása

A folyamatsablon kiindulásképp definiálja a módszertanhoz illeszkedő szabályrendszer szerinti elemeket. Azonban egy csapat működése soha nem ennyire merev. Valójában a megfelelő szabályrendszer és a hatékony működés kialakítása egy tanulási folyamat. Az idő előrehaladtával egyre világosabbá válik, hogy mi az, ami jól működik, és mi az, ami nem; melyek azok a mérőszámok, amik fontosak és relevánsak a csapat számára, és melyek azok, amelyek kevésbé.

A tanulási folyamat része a folyamatsablon, illetve azon belül a munkadarabok testreszabása. A leggyakoribb igény, ami felmerül egy csapat működése során, hogy a munkadarabot új mezővel vagy mezőkkel egészítsük ki.

A munkadarabok testreszabása trükkös feladat, de szerencsére ehhez is található megfelelő eszköz. Egy külön telepíthető komponensre, a Microsoft Visual Studio Team Foundation Server 2013 Power Tools csomagra van szükségünk. Ez a kiegészítő csomag a Team Foundation Serverre, illetve a fejlesztők által használt Visual Studio IDE mellé is telepíthető, és új képességekkel ruházza fel a szoftvercsomagot (B-3 ábra). A munkadarabok testreszabásának szempontjából ez a csomag kiemelt fontosságú, ugyanis ez biztosít grafikus felületet a munkadarabok testreszabásához.



B- 3 ábra:Team Foundation Server 2013 Power Tools telepítése során használható kiegészítések

> *Ennek a könyvnek az írásakor a munkadarabok testreszabási lehetőségét még csak a saját eszközökre telepített Team Foundation Server biztosítja, a Visual Studio Online mögött lévő csapatmunka eszköz azonban nem. Ennek a hiányzó képességnek a fejlesztése – a rendelkezésre álló információk alapján – folyamatban van.*

A csomag a telepítést követően a Visual Studio 2013 IDE kiegészítésével elérhetővé teszi a folyamatsablon szerkesztéséhez tartozó Process Editort, amely a teljes folyamat testreszabását teszi lehetővé.

A munkadarabok testreszabásának folyamata

Az egyes munkadarabok testreszabása vagy egy új típus elkészítése az ún. WIT (*Work Item Type*) definíciós fájl segítségével lehetséges. A testreszabás folyamata egyszerű lépésekből áll:

- **A WIT definíciós fájl kiexportálása.** Ez a lépés a definíciós állományról egy másolatot készít, amit szabadon szerkeszthetünk. A lokális példány módosítása önmagában nem módosítja a szerver viselkedését.
- **A definíciós file szerkesztése.** A WIT szerkesztő segítségével egy grafikus felületen szabhatjuk testre az egyes munkadarab típusokat.
- **A definíciós file importálása.** Ebben a lépésben a lokálisan szerkesztett munkadarabokat visszaimportáljuk a team projectbe. Innentől a mezők érvényesek lesznek, és az új munkadarabok mögött már az új folyamatok lépnek érvénybe. Fontos lehet ilyenkor végiggondolni, hogy a régi munkadarabokra ez milyen hatással lehet.

Az egyes munkadarabok szerkesztése három fő lépésből áll.

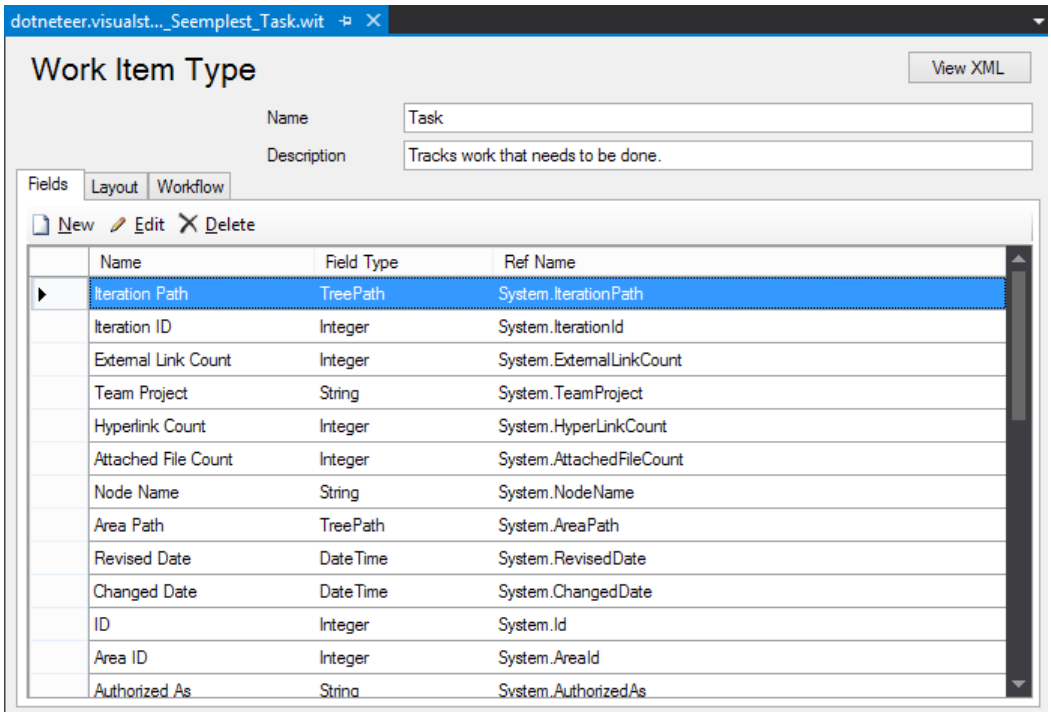
- A típus kiterjesztése mezők hozzáadásával
- A kapcsolódó felhasználói felület elrendezése
- A háttérben húzódó állapotgép módosítása

A típus kiterjesztése mezők hozzáadásával

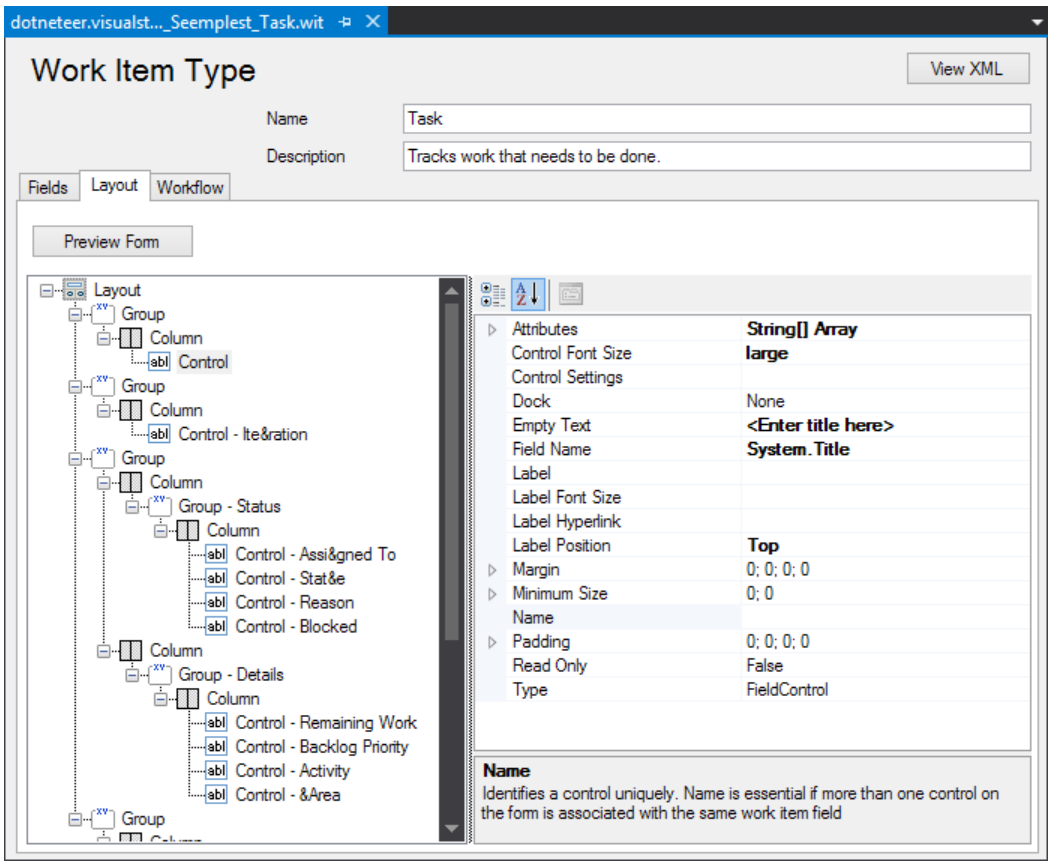
Ebben a lépésben egy új mezőt definiálunk. Meghatározzuk a mező nevét, és ami a legfontosabb, a típusát. A típus előre definiált, a Visual Studio által támogatott típusok közül kerül kiválasztásra. A mező testreszabásakor a típus hozzárendelésen túl különböző megkötéseket és értékhalmazokat is definiálhatunk. Így például meghatározhatjuk, hogy egy mező egy legördülő listából legyen kiválasztható. A lista tartalma lehet statikus, előre definiált, de lehet némi kódolás eredményeképpen előállítható dinamikus lista is, például felhasználók listája. A mezőhöz olyan egyéb attribútumok is csatolhatóak, mint például a kötelezőség, illetve alapértelmezett érték. A B-4 ábrán a Task definíciójához tartozó mezőket láthatjuk.

A kapcsolódó felhasználói felület elrendezése

A mezők definíciójának megtekintése önmagában nem kevés a testreszabáshoz. A felhasználói felület átalakítása, a munkadarab űrlapján egy-egy új mező elhelyezése szintén a gyakori tevékenységek közé tartozik. A felhasználói felületre saját vezérlőelem is elhelyezhető, ez azonban egy lényegesen bonyolultabb testreszabási lépés. Az esetek túlnyomó többségében az előre definiált, jól bevált vezérlők egyikéből választunk, és azt rendeljük az űrlaphoz. A B-5 ábra a felhasználói felület testreszabását biztosító – kicsit „fapadosnak” tűnő – felületet mutatja be, amelyen a Task típusú munkadarab felületének elrendezését szerkeszthetjük.



B- 4 ábra: Munkadarab típusok kiterjesztése mezőkkel

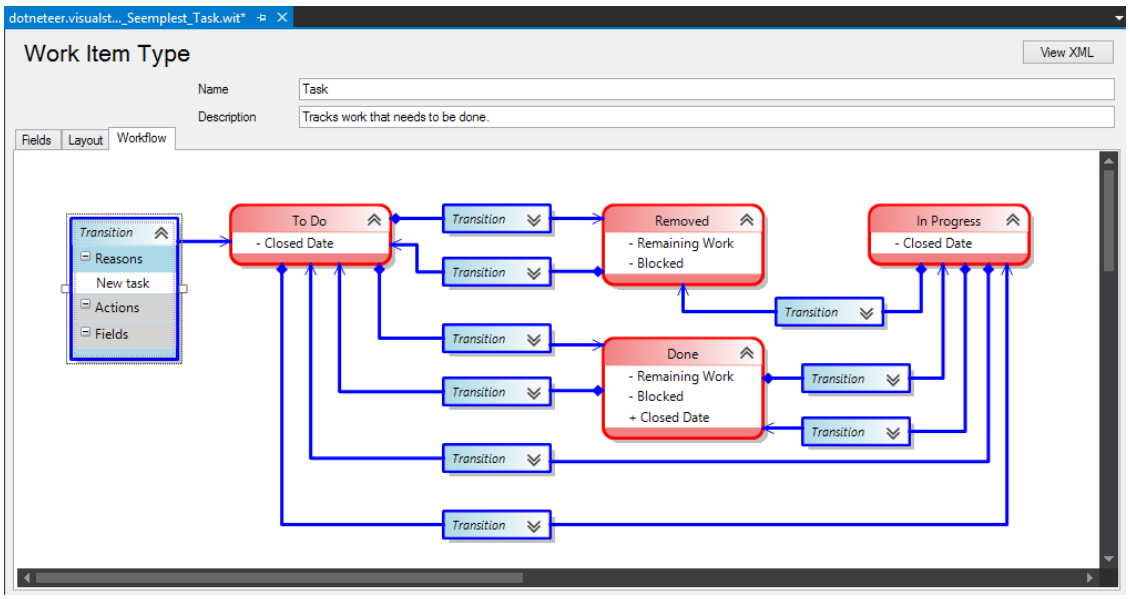


B- 5 ábra: A munkadarab űrlapjának testreszabása

A háttérben húzódó állapotgép módosítása

Amennyiben csak egy olyan egyszerű mezőt kívánunk definiálni, ami csupán adattárolási funkcionalitással rendelkezik, aránylag egyszerű dolgunk van, ugyanis, az első két lépés után véget is ér a munkánk. Azonban ha a mező folyamatok követésére is képes – például állapotot tárol, és az állapotváltozást egy jól definiált állapotgép határozza meg, ahol az átmenetek különböző feltételekhez vannak kötve, akár

jogosultsági megszorításokkal kiegészítve –, jóval izgalmasabb feladat elé nézünk. A munkadarab mögött egy munkafolyamat (*workflow*) rejlik, amely egy állapotgép segítségével testreszabható. A B-6 ábrán a Task típusú munkadarab mögött húzódó állapotgépet láthatjuk. A piros dobozok az állapotokat definiálják, a kék dobozok az állapotok közötti átmeneteket reprezentálják. Az átmeneteken látható, hogy további attribútumok is definiálhatóak, amelyek az átmenetek aktiválásához járulnak hozzá.



B- 6 ábra: A Task típus mögötti állapotgép

Sajnálatos módon nem minden megkötés, attribútum, illetve funkció definiálható ezeken a felületeken. Az egész WIT-et egyetlen XML dokumentum írja le. Ennek a dokumentumnak a közvetlen szerkesztésével minden opciót kihasználhatunk, például állapotváltást jogosultsághoz köthetünk.

Összegzés

A munkadarabok szerves részét képezik a Team Foundation Server-en végzett csapatmunkának. Ezeknek az egységeknek a segítségével vagyunk képesek a munkát koordinálni, felmérni, becsülni. A munkadarabok által biztosított információk nagymértékben hozzájárulnak a követhetőséghez, valamint az iterációról iterációra történő tanuláshoz, fejlődéshez. A fejlődés része az a felismerés is, hogy bizonyos esetekben a megfelelő követhetőséghez és koordinációhoz a munkadarabokat újabb és újabb tulajdonságokkal, a sablonokat újabb típusokkal kell kiegészíteni. A munkadarab módosításához rendelkezésre álló eszközök segítségével relatíve könnyen saját igényeinkhez igazíthatjuk az egyes típusokat és folyamatokat.