Visual Studio

Agilis termékfejlesztés

Hogyan növeld alkalmazásod értékét a Visual Studio 2015 segítségével.

Bevezetés

Egy jó alkalmazás kifejlesztéséhez szükség van jó programozókra. Ahhoz, hogy ebből értékes alkalmazás legyen, a programozáson túlmutató dologra van szükség: alkalmazásodat termékként kell kezelned, folyamatosan szem előtt tartva az alábbiakat:

Alkalmazásod felhasználóknak készül. Meg kell értened — ki kell találnod — ők miért és hogyan használják — vagy hogyan szeretnék használni — alkalmazásodat.

Értéket kell kínálnod! Mindig valamilyen cél elérésében kell segíteni mindazokat, akik az alkalmazásodat használják. Annak olyan valós problémákra kell megoldást adnia, amellyel felhasználói gyakran szembesülnek.

Alkalmazásodnak frissnek kell maradnia! A felhasználóid és a környezet változásait folyamatosan követned kell! Figyelj a visszajelzésekre, használd fel ezeket, hogy a jövőben még több értéket tudj kínálni!

Minőségre kell törekedned! Ahogyan te is általában jó kiszolgálást, adott minőséget vársz el egy fodrásztól vagy kedvenc éttermedtől, kocsmádtól — ugyanezt várják el azok, akik az alkalmazásoddal találkoznak.

Gyorsnak kell lenned! Felhasználóid türelmetlenek. Nem fognak heteket vagy hónapokat várni egy számukra fontos termékképességre, egyszerűen keresnek egy másik terméket, ha a tied már nem biztosít elegendő értéket számukra.

Tanulnod kell a tapasztalatokból! Minden felhasználói visszajelzést, hibáidat, sikereidet és megfigyeléseidet fel kell használnod, hogy azokból tanulva javíthasd, értékesebbé tedd az alkalmazást!

Furcsának tűnhet számodra, hogy egy olyan dokumentumban szembesülsz ezekkel az állításokkal, amelyek a Visual Studióról szólnak. Ez nem véletlenül van így! A Visual Studio fejlesztőknek és termékcsapatoknak készül — és folyamatosan szem előtt tartja azokat az elvárásokat, amelyeket a fentiekben az alkalmazások értékei kapcsán mutattunk be.

Nagyon sok eszközt használhatsz céljaid elérésére — vagyis a fenti szemléletmódot követő alkalmazások készítésére —, ezek közül egy a Visual Studio. Ebben a dokumentumban bemutatjuk, hogyan segíthet téged és csapatodat értékes, a valódi alkalmazások fejlesztésében, termékként való kezelésében az *agilis szemléletmód* — és ehhez mivel tud hozzájárulni a *Visual Studio*.

Miért beszélünk termékről?

Ha nem egyszeri — „eldobható” — alkalmazást készítesz, amelyet csak egy szűk csoport használ meglehetősen korlátos ideig, lehet, hogy azt termékként kellene kezelned. Ha az alkalmazás célja nem pusztán annak elkészítése, hanem az, hogy valódi felhasználók valódi problémákat oldjanak meg vele, biztos lehetsz benne, hogy egy termékkel állsz szemben, még akkor is, ha annak csak nagyon kevés funkciója van.

Miért beszélünk agilitásról?

Mint szoftverekkel dolgozó szakember, te is észrevehetted, hogy a legtöbb szoftver már gyakran hetente, havonta kínál és telepít újabb módosításokat — a jobbak mindezt már úgy végzik a háttérben, hogy észre sem veszed őket. A felgyorsuló világban úgy kell egyre gyorsabban, egyre rövidebb ciklusokban szoftvermódosításokat kibocsátanunk, hogy eközben a minőségi elvárások folyamatosan nőnek. Ezt csak átmeneti ideig lehet a fejlesztői hatékonyság — kódolói ügyesség és termelékenység — irányából gyorsítani. Az igazi lehetőséget az jelenti, ha csillagháborús termékképességek helyett kisebb, átgondoltabb és értékesebb termékbővítményeket tudunk folyamatosan a felhasználók kezébe adni.

Az agilis szemléletmód erről a működési modellről szól:

Bevezetés

Átgondolt módon választjuk ki azt a néhány termékképességet, amelyet leszállítani tervezünk a következő ciklusban.

Jó minőségben, ellenőrizhető módon készítjük el és szállítjuk le ezeket a képességeket.

„Éhezünk” a visszacsatolásra, azokat azonnal beépítjük a termék kezelésébe, hogy a következő ciklusban a megfelelő képességeket szállítsuk le.

Folyamatosan csökkentjük műszaki adósságainkat, fejlesztjük a csapatmunkát.

1. Agilis termékkezelés

Nagyon sok szoftverfejlesztési projekt fullad kudarcba, vagy legalább is erőteljesen megoszlanak a vélemények abban a tekintetben, hogy azok sikeresek-e. A problémák látható része a költségek túllépése, az időkeretből való kicsúszás, de gyakran más nehézségek is társulnak ezekhez.

Sok elemzés, tanulmány készült az okokról. Ebben a dokumentumban nem szeretnénk a részletekbe belemenni — könyvtárpolcokat lehetne megtölteni a kapcsolódó művekkel. Érdemes viszont megjegyezni az alábbia állítást, amelyre a legtöbb tanulmány rámutat:

Egy szoftver fejlesztése sokkal könnyebben és gyakrabban bukik el a termék és a megvalósításához szükséges feladatok kezelése során, mint a technológián és annak kockázatain. Gyakran rossz termékképességek vagy rossz időben készülnek el.

Termékképesség és érték

Nehéz hosszú, felsorolásszerű specifikáció segítségével terméket fejleszteni. Még ha egy-egy ilyen specifikáció prioritásokkal segíti is a fontos és a kevésbé fontos dolgok szétválasztását, ritkán ad támpontokat a fejlesztőknek abban a tekintetben, hogy melyik képességek fejlesztésével érdemes a termékfejlesztési projekt egy adott pontján foglalkozni.

Az agilis szemlélet lényege, hogy rövid idő alatt, viszonylag kisméretű termékképességeket igyekszünk leszállítani. Ezzel az a célunk, hogy gyorsan visszacsatolást kapjunk, és szükség esetén módosíthassuk az adott termékképességet — vagyis már a fejlesztés közben is tanuljunk.

Ahhoz, hogy ezt megtehessük, a termék egészét — amely önmagában valószínűleg kezelhetetlenül nagy — sokkal kisebb képességekre bontjuk, egészen addig, amíg azok már önmagukban kezelhetővé válnak. Az így lebontott termékképességekhez értékét rendelünk — ez segít abban, hogy alapvetően az értékes dolgokra fókuszálhassunk, az értéktelen termékképességek ne zavarják meg gondolkodásunkat.

Termékgazda

A termékek képességeinek kezeléséhez kapcsolódó szerepkört az agilis módszerek és keretrendszerek általában *termékgazdának* —angolul *product owner*nek—nevezik. Az ebben a szerepkörben lévő csapattagnak olyan ismeretekre van szüksége, amelyekkel leginkább üzleti elemzők, termékmenedzserek, illetve gyakran projektmenedzserek rendelkeznek. Ő az, aki a csapat segítségével kialakítja a termékképesség jellemzőit, meghatározza azok értékét, válaszokat ad a felmerülő kérdésekre, segíti a csapatot az adott képesség megértésében, kialakításában.

Product Backlog

A termékekhez kapcsolódó feladatok az ún. *product backlog*ba kerülnek. A termékgazda rendszerezi az itt található feladatokat: azok tartalmát finomítja, sorrendjét változtatja, a nyitott kérdésekre megkeresi a válaszokat.

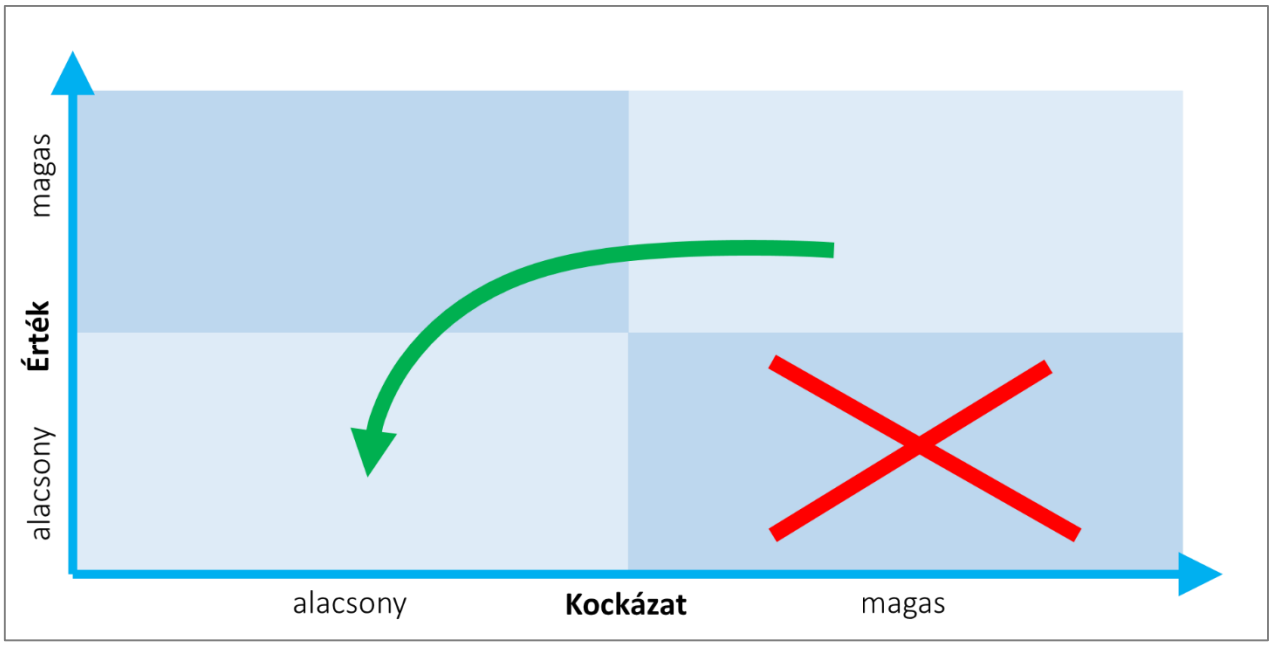
Megvalósítási sorrend

A termék képességeinek összegyűjtése után előbb-utóbb azokat meg is kell valósítania a csapatnak. Az agilis termékkezelés a megvalósítás sorrendjének kialakítása során az 1-1 ábrán látható mátrixot használja. Ebben az összegyűjtött képességeket négy részre bontjuk, azok értéke (alacsony, magas) és kockázata (alacsony,

magas) szerint. Az agilitás az értéket fontosnak tartja, ezért a magas értékű képességek kialakításával kezdünk. Azok közül is a magas kockázatúakat vesszük előre, hogy minél több idő maradjon még a tervezett projekt végéit a kockázatok kezelésére — vagy akár arra, hogy a túl nagy kockázati szint miatt a projektet leállíthassuk, mielőtt még jelentősebb erőforrásokat égetnénk el.

Ezt követi a magas értékű és alacsony kockázatú, majd az alacsony értékű és alacsony kockázatú képességek megvalósítása.

Az agilis projektek a magas kockázatú, ám alacsony értékű termékképességeket általában nem valósítják meg. Miért is akarnánk olyan funkciót kifejleszteni, amely sok kockázattal jár, de ha elkészül, akkor is csak kis értékkel járul hozzá a termékhez? Talán valami hasznosabb képességre is fordíthatnánk az így eltöltött időt!



1-1 ábra: A termékképességek megvalósítási sorrendje

Mintapélda

Tegyük fel, hogy egy kis csapat olyan alkalmazást szeretne elkészíteni, amely segítségével annak felhasználói ellenőrző listákat kereshetnek — például utazások szervezéséhez (milyen teendőim vannak?), gyermekvárásra felkészülve (hogyan is kell berendezni egy gyerekszobát?), vagy éppen egy búvártúrára indulás előtt (milyen felszerelést és eszközöket kell magammal vinnem?) —, és azokat fel is használhatják.

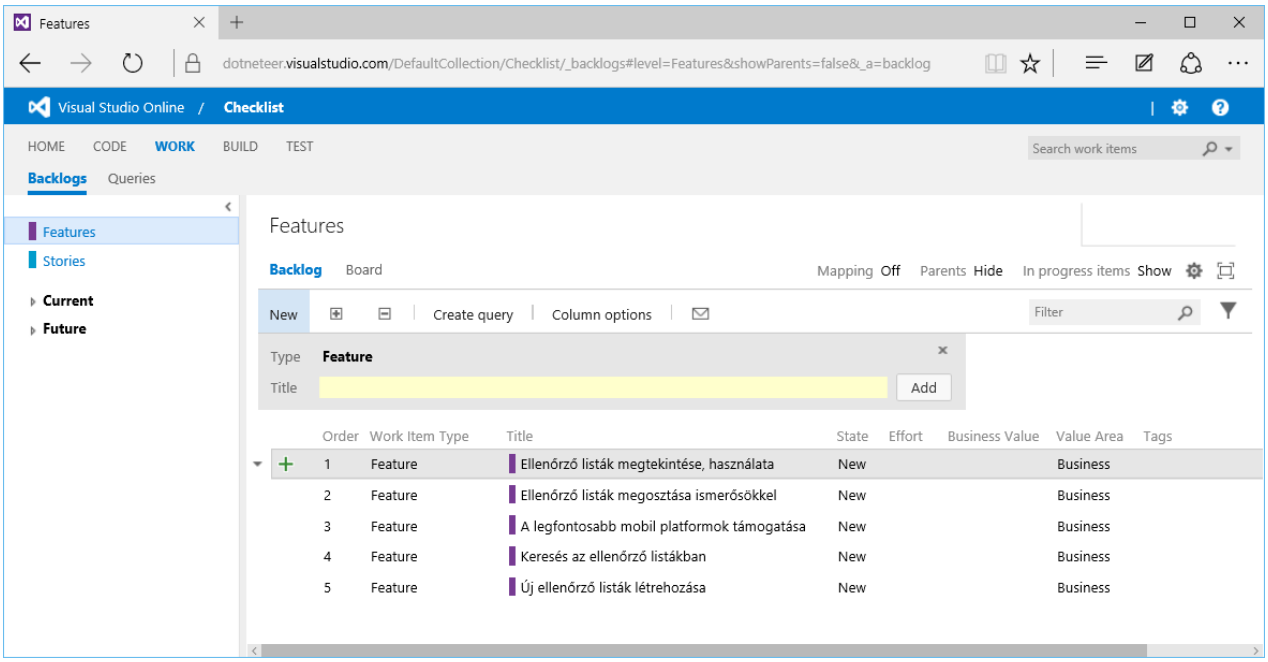
A csapat lehetőségei nagyjából három hónap időtartamra fedezik a fejlesztés költségeit, a potenciális szponzor egy hónap múlva szeretne egy demót látni. A szponzor akkor biztosít további támogatást, ha az alkalmazásban már a demó után látja azt a lehetőséget, hogy nagy felhasználói bázist tud majd a segítségével kiépíteni.

Javasolt termékképességek

A csapat termékgazdája Tibor, aki a szponzorral és a csapattal beszélgetve kialakította az alkalmazás legfontosabb termékképességeit. A fejlesztőcsapat háromfős (Anita, József és László), mindegyikük néhány év tapasztalattal, különböző technológiai háttérrel rendelkezik. Közösen úgy döntenek, hogy a demó kifejlesztését a Visual Studio Team Services[1] (továbbiakban: VSTS) eszközeivel, és a letöltött Visual Studio Professional 2015-tel kezdik majd el.

[1] Korábbi nevén: Visual Studio Online

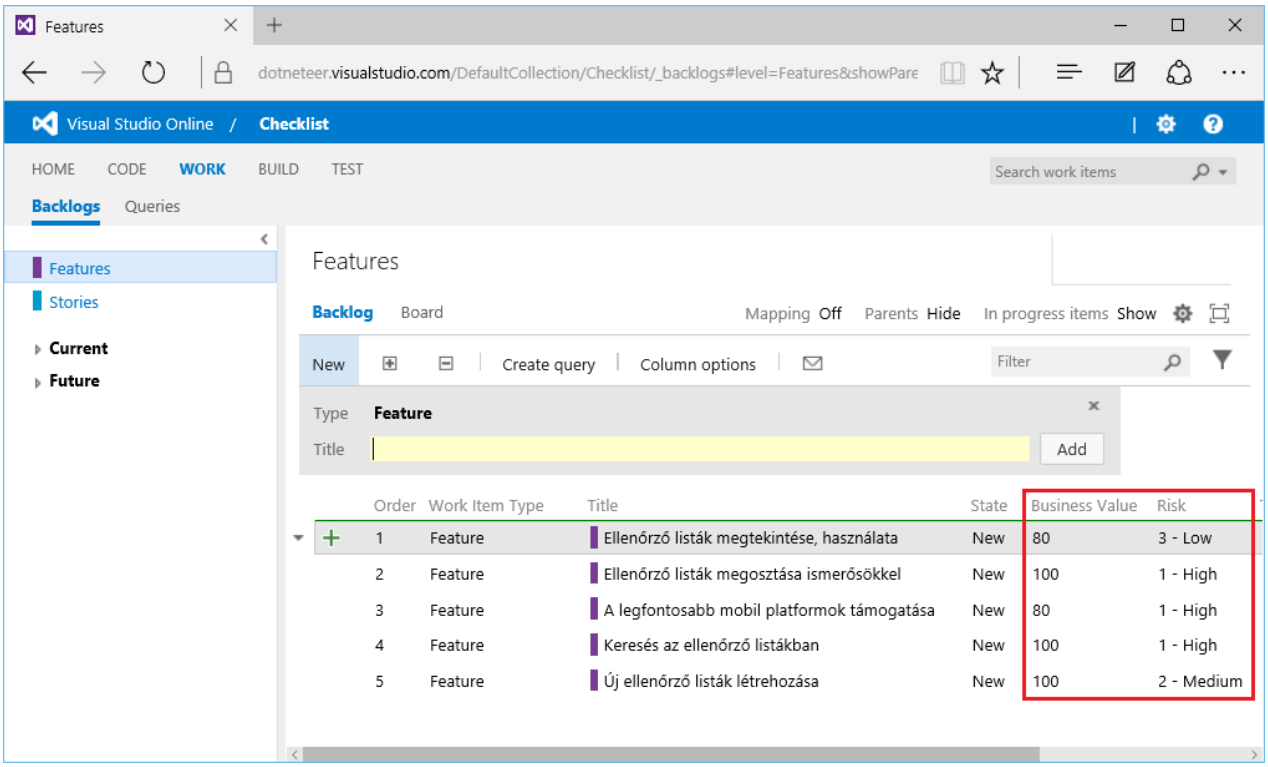
Tibor már létrehozta a VSTS projektet, és az egyeztetések alapján összegyűjtötte az általa legfontosabbnak tartott termékképességeket (1-2 ábra).



1-2 ábra: A tervezett termékképességek

Értékek és kockázatok

Tibornak van egy elképzelése arról, hogy mely képességek milyen értékkel bírnak, illetve milyen kockázattal járnak, ezt végig beszélte a csapattal, és közösen módosították a fenti listát (1-3 ábra).

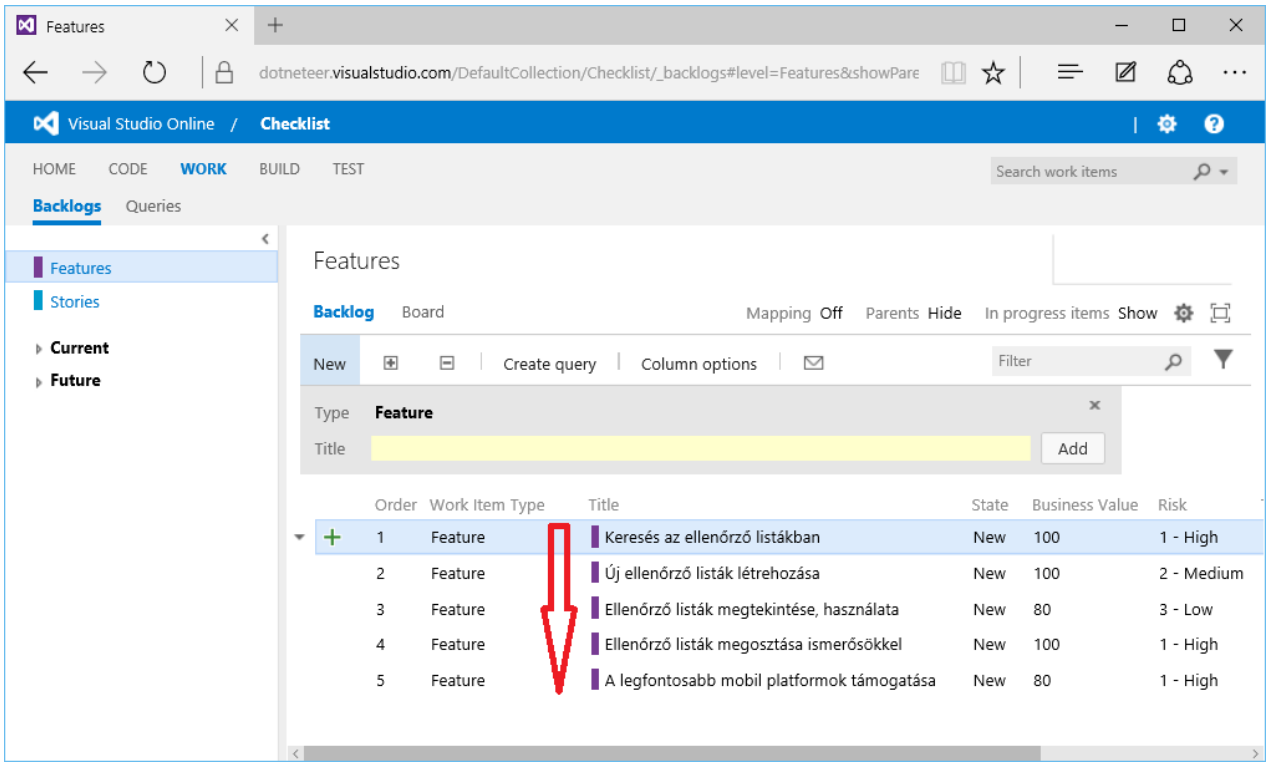


1-3 ábra: Kockázatok és értékek

A csapat úgy döntött, hogy 1-100 skálán adja meg egy-egy termékképesség értékét, a 60 és annál nagyobb értékeket tekintik magasnak. A kockázat értékét három szinten definiálták (High, Medium, Low — ahogyan azt a VSTS kínálja fel), az első két kategóriát tekintik magasnak (az 1-1 ábrán lévő mátrixba sorolásnál).

A csapat felismerte, hogy több termékképesség prioritása és értéke is túl nagy, ezért sorba rendezték és átalakították a listát, hogy egy jó és értékes demó változatot tudjanak az első hónap végére kialakítani.

Előre vették a demó szempontjából legfontosabb képességet, a keresést. Úgy döntöttek — a felhasználói bázis növelése céljából —, hogy egyelőre minden felhasználó létrehozott ellenőrző listája publikus lesz, és majd a későbbiekben teszik azt zárhatóvá, megoszthatóvá. A demóhoz nincs szükségük arra, hogy a legfontosabb mobil platformokat támogassák, ezért egyelőre csak egy webes alkalmazást készítenek, amely adaptív felhasználói felülettel fog működni, így könnyen demonstrálható annak mobil nézete (1-4 ábra).



1-4 ábra: A termékképességek priorizálva

A csapat már látja a legfontosabb előtte álló termékképességeket, de még közelebb kell kerülnie ahhoz, hogy a megvalósításhoz hozzákezdhessen.

2. Backlog kezelés

A csapat számára már világos az, hogy a demó megvalósítását az ellenőrző listákban való kereséssel kell kezdenie, majd utána az új ellenőrző listák létrehozása felé kell haladnia. Ez azért is fontos, mert a szponzor valószínűleg olyan demót szeretne látni, amely meggyőzi őt az alkalmazás értékéről. Márpedig egy ilyen alkalmazást a felhasználók jelentős része arra fog használni, hogy a saját teendői kapcsán keressen a már meglévő ellenőrző listák között.

A csapat számára ezen a ponton az jelenti a legnagyobb kihívást, hogy még nem tudják, pontosan mit is kell érteni a „Keresés az ellenőrző listákban" termékképesség kapcsán, így nehezen tudják megmondani, hogy ez egészében belefér-e abba az időbe, amely a demóig rendelkezésükre áll.

Sztori létrehozása

Azért, hogy a termékképességet a csappattal alaposan át lehessen beszélni, Tibor — a termék gazdája — egy ún. *user story*t (a későbbiekben ezt csak sztorinak nevezzük) készít:

Látogatóként keresni szeretnék a rendszerben lévő ellenőrző listák között azért, hogy megtaláljam és megtekinthessem azokat, amelyek a tevékenységeimben segítenek.
Leírás: A kereséshez egyetlen szövegmezőbe beírt kulcsszót (akár több részből is állhat) használunk, a webes keresőkhöz hasonlóan. *Elfogadási kritériumok:* A keresés az ellenőrző lista címében és tételeiben is megtalálja a kulcsszót. A keresés nem érzékeny a kis- és nagybetűkre. A keresés nem érzékeny az ékezetekre. A keresés szótöredékekre is működik, kezelni tudja a képzett szavakat (jelzők, ragok stb.). A keresés angol és magyar nyelven is működik. A találati listában előbb jelennek meg azok a találatok, ahol az ellenőrző lista címében van a keresett kulcsszó. A találati listában előbb jelennek meg azok az ellenőrző listák, amelyeket több felhasználó tekintett meg. A találati listában látható az ellenőrző lista címe, illetve azoknak a tételeknek a rövid kivonata, amelyekben a kulcsszó megtalálható. A kulcsszó előfordulásai vizuálisan ki vannak emelve. A találati lista egy elemére kattintva az adott ellenőrző lista részleteinek megtekintésére jut a felhasználó. *Demó forgatókönyv:* #TBE

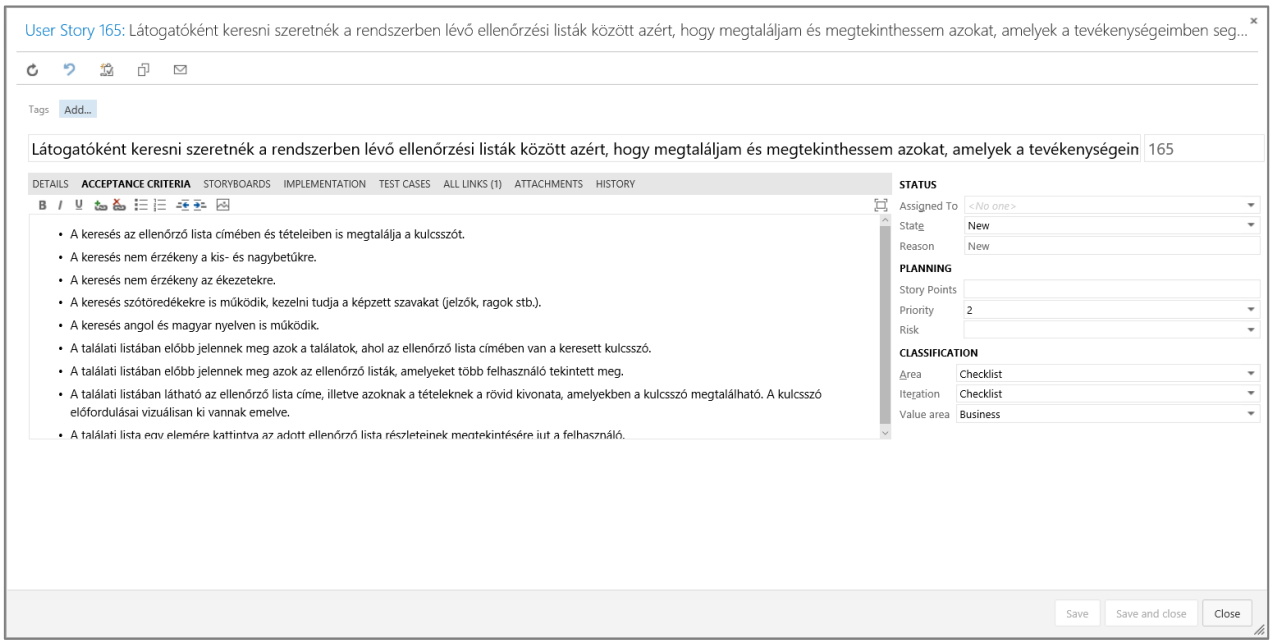
Ebben a sztoriban Tibor olyan címet használ, amely elmondja, hogy *ki* (látogató), *mit* (keresés) és *milyen célból* (értékes ellenőrző lista megtalálása) vár el az alkalmazás adott képességétől. Tibor tudja, hogy ezek a jellemzők fontosak ahhoz, hogy a csapat meg tudja érteni a sztorit, különösen a felhasználási célra igyekszik pontos megfogalmazást adni. Ha nem így tenne, gyengítené a csapatot, megfosztaná őket a kreativitásuktól: a cím „milyen célból" része segíti a csapatot abban, hogy a termékképesség értékére fókuszálhasson.

A sztorihoz Tibor rövid leírást is ad, amelyben tisztázza, hogy egyszerű kulcsszavas keresésről van szó.

Ahhoz, hogy a csapat pontosan tudja, milyen elvárásoknak kell megfelelnie, Tibor tételesen összegyűjtötte az elfogadási kritériumokat. Azt, hogy a csapat megfelelően valósítotja-e meg a feladatot, az elfogadási kritériumok ellenőrzésével — azok tényleg teljesülnek? — lehet majd leellenőrizni.

Az ellenőrzést a csapat azzal segít egyszerűbbé és átláthatóbbá tenni, hogy demó forgatókönyvet definiál, amely segít annak tisztázásában, milyen demonstrációval igazolják azt, hogy a termékképesség az elvárt módon készült el. Tibor egyelőre még nem készített ilyen forgatókönyvet — ezért használta a #TBE jelölést, amelyet a csapat minden tagja ért —, ehhez a többiek segítségére lesz majd szüksége.

Amint azt a 2-1 ábra is mutatja, ezt a sztorit Tibor a Checklist projekt product backlogjában rögzítette.



2-1 ábra: A sztori kártyája

A sztori elemzése

Tibor bemutatja a csapatnak a sztorit, azt közösen megvitatják. A csapat észrevételeit rávezetik a sztori kártyájára:

Látogatóként keresni szeretnék a rendszerben lévő ellenőrző listák között azért, hogy megtaláljam és megtekinthessem azokat, amelyek a tevékenységeimben segítenek.
Leírás: A kereséshez egyetlen szövegmezőbe beírt kulcsszót (akár több részből is állhat) használunk, a webes keresőkhöz hasonlóan. *Elfogadási kritériumok:* A keresés az ellenőrző lista címében és tételeiben is megtalálja a kulcsszót. A keresés nem érzékeny a kis- és nagybetűkre. A keresés nem érzékeny az ékezetekre. *#HARD* A keresés szótöredékekre is működik, kezelni tudja a képzett szavakat (jelzők, ragok stb.). #RISKY A keresés angol és magyar nyelven is működik. A találati listában előbb jelennek meg azok a találatok, ahol az ellenőrző lista címében van a keresett kulcsszó. #??? A találati listában előbb jelennek meg azok az ellenőrző listák, amelyeket több felhasználó tekintett meg. A találati listában látható az ellenőrző lista címe, illetve azoknak a tételeknek a rövid kivonata, amelyekben a kulcsszó megtalálható. A kulcsszó előfordulásai vizuálisan ki vannak emelve. A találati lista egy elemére kattintva az adott ellenőrző lista részleteinek megtekintésére jut a felhasználó. *Demó forgatókönyv:* #TBE

A csapat a saját megegyezésükön alapuló jelöléseket — #HARD, #RISKY, #??? — használ a nehezen megvalósítható, kockázatos, illetve további kérdéseket felvető kritériumok megjelölésére. Amíg ezekkel nem kezdenek valamit, nem ismerik elég pontosan a sztorit, így nem tudnak jó becslést adni arra, hogy annak megvalósítása milyen feladatokból fog állni, és várhatóan mennyi ráfordítást igényel.

A sztori felbontása

Az egyeztetések során a csapat megállapítja, hogy ebben a formában a feladat túl nagy és kockázatos ahhoz, hogy a demóig elkészüljön, ezért ezt a nagy sztorit az elfogadási kritériumok és a kockázatok mentén több kisebb sztorira bontják szét. Az alábbi sztorikat hozzák létre (itt most csak azok címeit olvashatod):

Látogatóként keresni szeretnék a rendszerben lévő ellenőrző listák között azért, hogy megtaláljam és megtekinthessem azokat, amelyek a tevékenységeimben segítenek.

Látogatóként a rendszerben lévő ellenőrző listák között szótöredékekre is szeretnék keresni azért, hogy olyan ellenőrző listákat is megtaláljak, amelyek az eddigi keresések során nem szerepeltek a találati listában.

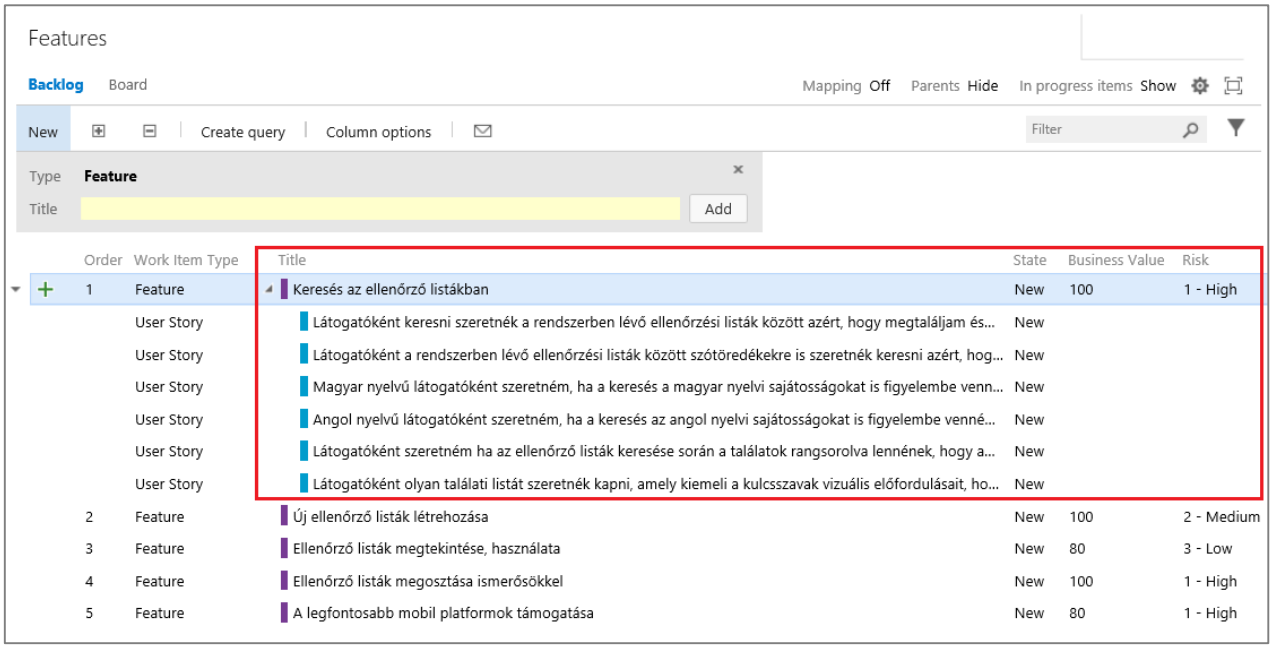
Magyar nyelvű látogatóként szeretném, ha a keresés a magyar nyelvi sajátosságokat is figyelembe venné az ellenőrző listák keresésnél, hogy jobb találati listát kaphassak.

Angol nyelvű látogatóként szeretném, ha a keresés az angol nyelvi sajátosságokat is figyelembe venné az ellenőrző listák keresésnél, hogy jobb találati listát kaphassak.

Látogatóként szeretném, ha az ellenőrző listák keresése során a találatok rangsorolva lennének, hogy a számomra fontosabbakat hamarabb találjam meg a találati listában.

> Látogatóként olyan találati listát szeretnék kapni, amely vizuálisan kiemeli a kulcsszavak előfordulásait, hogy a hosszabb listákat gyorsabban tudjam áttekinteni.

Tibor ügyel arra, hogy a Checklist projekt kezelésénél az új sztorikat is a „Keresés ellenőrző listákban" termékképesség alá sorolja be (2-2 ábra).



2-2 ábra: A termékkéesség és a hozzátartozó sztorik

Demó forgatókönyv készítése

Az eredeti sztori így már rövidebb lett, a csapat azt önmagában megvalósíthatónak tartja. Azért, hogy a fejlesztési és tesztelési folyamatot segítség az elvárásokra fókuszálni, illetve hatékonyan készülhessenek fel a demóra, a sztorikhoz demó forgatókönyveket rendelnek. Az első (és azóta már kisebb részekre darabolt) sztori végleges formája így néz ki:

Látogatóként keresni szeretnék a rendszerben lévő ellenőrző listák között azért, hogy megtaláljam és megtekinthessem azokat, amelyek a tevékenységeimben segítenek.

Leírás:

A kereséshez egyetlen szövegmezőbe beírt kulcsszót (akár több részből is állhat) használunk, a webes keresőkhöz hasonlóan.

Elfogadási kritériumok:

A keresés az ellenőrző lista címében és tételeiben is megtalálja a kulcsszót.

A keresés nem érzékeny a kis- és nagybetűkre.

A keresés nem érzékeny az ékezetekre.

A találati lista egy elemére kattintva az adott ellenőrző lista részleteinek megtekintésére jut a felhasználó.

Demó forgatókönyv:

A sztori demójához egy demó/teszt adatbázist hozunk létre, azt előre feltöltjük egy szöveges teszt állományból. Az ellenőrző lista részletes nézete egyelőre csak egyszerűen megjeleníti a lista címét és elemeit.

1. Rákeresek a „hasra esés" kulcsszóra: üres találati listát kapok.
2. Rákeresek az „árvíztűrő tükörfúrógép" kifejezésre: 2 találatot kapok, egyet a címben, egyet egy ellenőrző lista tételében.
3. Rákeresek az „arvizturo tukorfurogep" kifejezésre: 2 találatot kapok, egyet a címben, egyet egy ellenőrző lista tételében, pontosan úgy, mint az előző keresési lépésben.
4. Rákeresek a „ScRUm" kifejezésre: 4 találatot kapok, ezekben szerepelnek a „scrum", „Scrum" és „SCRUM" szavak egyaránt. Megmutatom, hogy egy találatra kattintva az adott ellenőrző lista részletes nézetébe jutok.

Látható, hogy a demó forgatókönyv kialakításánál arra törekedett a csapat, hogy ezt a sztorit minden más egyéb sztoritól függetlenül meg tudja valósítani — és természetesen annak működését le is tudja ellenőrizni. Bár sok csapat esetleg először egy olyan sztorit valósítana meg, amellyel ellenőrző listákat lehetne létrehozni — azt gondolva, hogy ez előfeltétele a keresésnek —, erre nincs szükség: a csapat egy előre elkészített teszt adatbázist fog használni.

Hasonló módon, a sztori működésének igazolásához egyelőre egyszerű felület is elegendő az ellenőrző lista részleteinek megtekintéséhez — a csapat egy másik sztoriban definiálja majd azt, hogyan is kell majd ennek a képernyőnek kinéznie, működnie.

A backlog elemeinek priorizálása

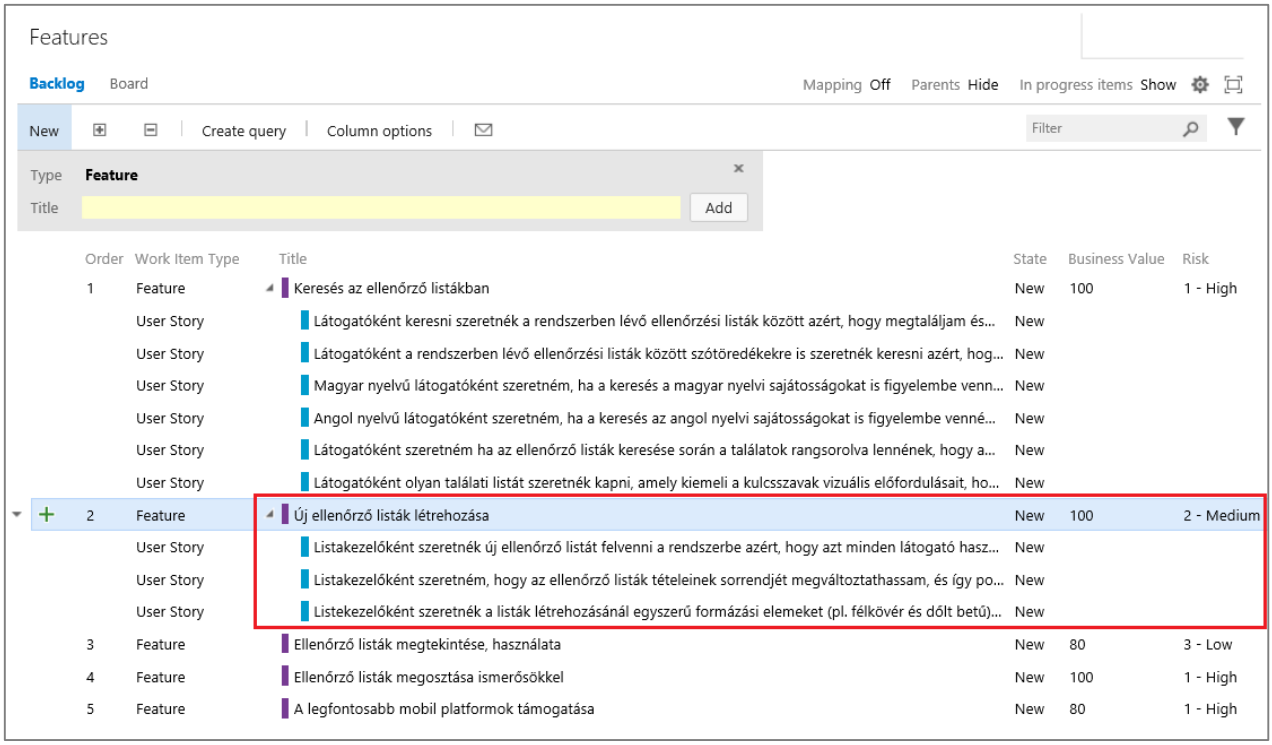
Tibor további sztorikat egyeztet a csapattal, és az alábbiakat veszi még fel a backlogba:

Listakezelőként szeretnék új ellenőrző listát felvenni a rendszerbe azért, hogy azt minden látogató használhassa.

Listakezelőként szeretném, hogy az ellenőrző listák tételeinek sorrendjét megváltoztathassam, és így pontosabb listákat készíthessek.

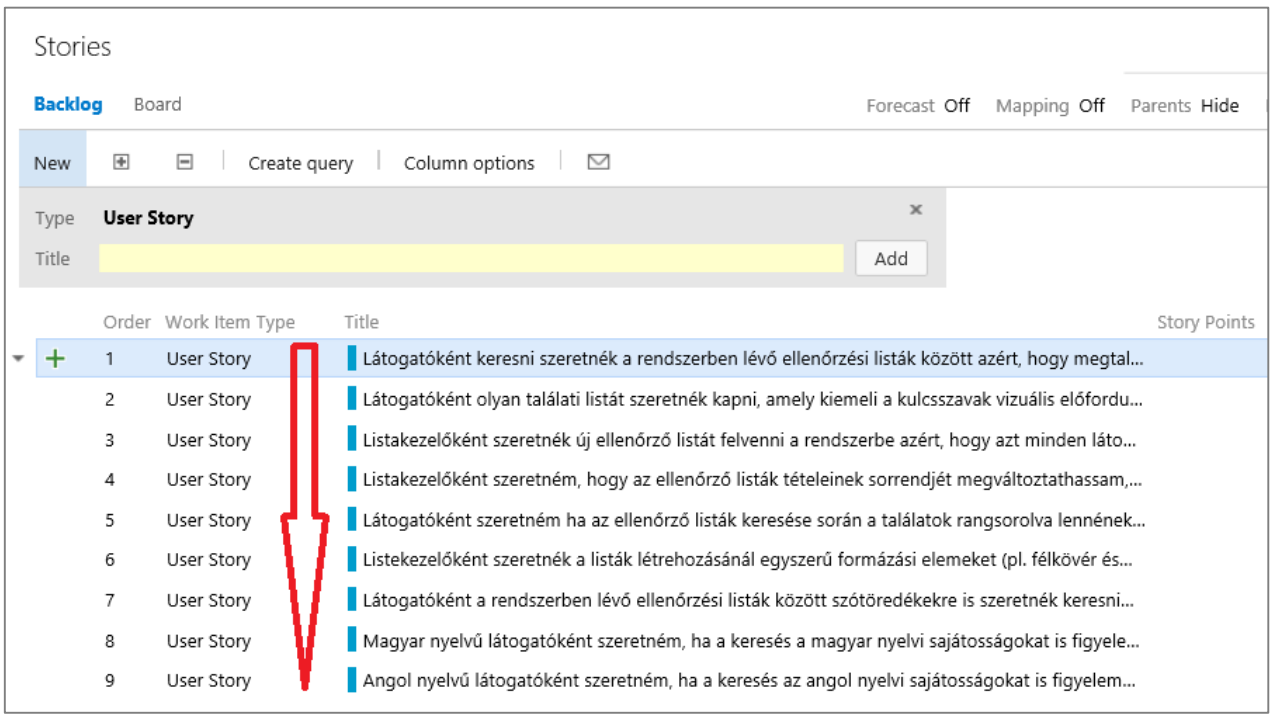
Listakezelőként szeretnék a listák létrehozásánál egyszerű formázási elemeket (pl. félkövér és dőlt betű) használni, hogy jobban áttekinthető és használható listákat tudjak létrehozni.

Értelemszerűen ezeket a sztorikat Tibor az „Új ellenőrző listák létrehozása" termékképességhez rendeli (2-3 ábra).



2-3 ábra: Új sztorik

Tibor a projekt portáljában sorrendezi — egyszerűen azok átmozgatásával — az eddig elkészült sztorikat, amint az a 2-4 ábra mutatja. A lista elején a magas értékű és kockázatú sztorik találhatók. A csapat számára ez a sorrend fontos jelzés: segíti őket abban, hogy az adott sorrendhez illeszkedő megvalósítási lehetőségekben gondolkodjanak.



2-4 ábra: A sztorik sorrendezés után

A csapat már közel jár ahhoz, hogy a megvalósításhoz hozzákezdhessen, de előbb még meg kell becsülnie a sztorik méretét.

3. Agilis fejlesztőkörnyezet kialakítása és deszkamodellek készítése

A csapatnak még három napja van arra, hogy megbecsülje az első néhány sztori méretét, majd hozzákezdjen annak megvalósításához. A csapat több webes technológiát is ismer és használ. Mivel nagy gyakorlatuk van az ASP.NET MVC használatában, valamint sok JavaScript alapú könyvtárat is rendszeresen használnak, úgy döntenek, hogy ASP.NET MVC 5-tel és Angular frameworkkel dolgoznak.

Már korábban is hallottak a TypeScript programozási nyelvről, és egy-két képességét ki is próbálták. Úgy döntenek, hogy az ellenőrző lista termék elkészítésénél ezt a programozási nyelvet is felveszik eszköztárukba.

Miért van szükség deszkamodellekre?

A rendelkezésre álló három napot arra szeretnék felhasználni, hogy néhány „deszkamodell"" (*proof-of-concept*) segítségével jobban megismerjék az eszközöket, áttekinthessék azok előnyeit és kockázatait, pontosabb képet kapjanak a sztorik megvalósításához szükséges erőforrások becsléséhez.

A csapat úgy dönt, hogy az alábbi feladatokra készít deszkamodelleket, hogy azokat közösen elemezve technológiai döntéseket hozhassanak, illetve információt nyerjenek a becslésekhez:

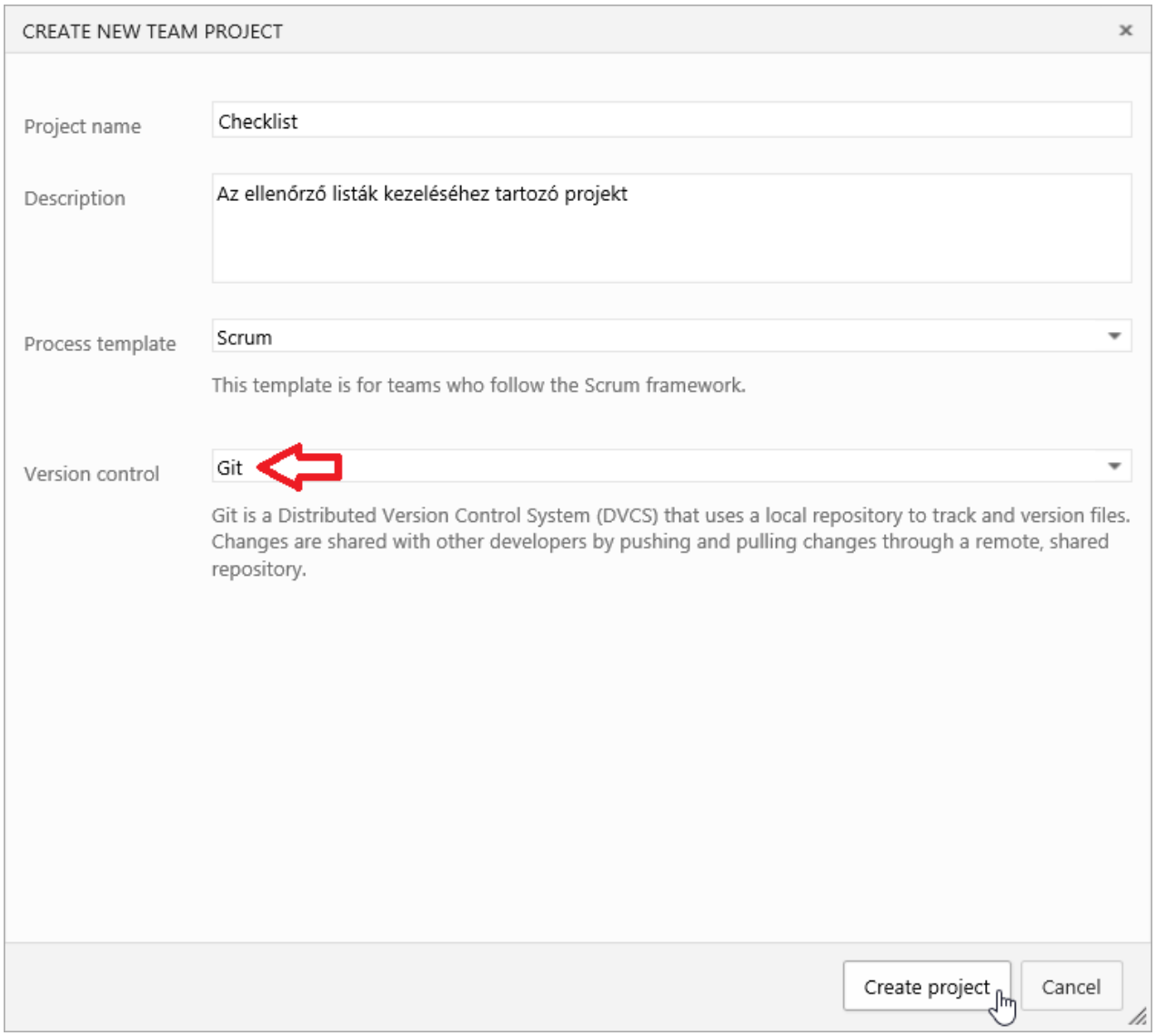
Egyszerű ellenőrzőlista megjelenítése ASP.NET 5 WebApi háttérszolgáltatással és Angular frontenddel — TypeScript használatával (József)

Egyszerű keresés 100 000 generált ellenőrző listában MongoDb adatkezeléssel (Anita)

Egyszerű keresés 100 000 generált ellenőrző listában SQL Server adatkezeléssel (László)

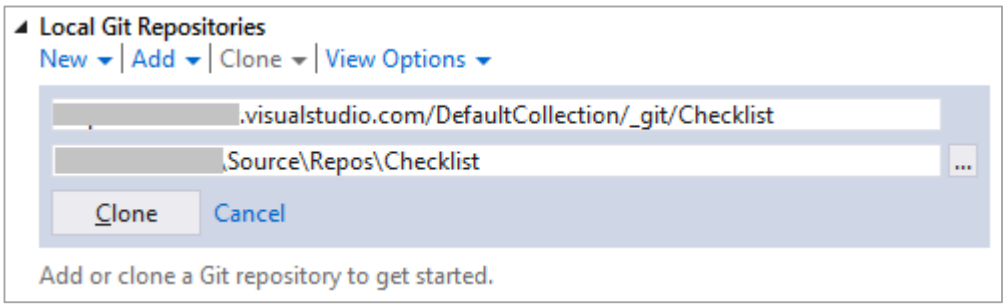
A verziótár kialakítása

A projekt létrehozásakor nemcsak a product backlogot, de a verziótárat is létrehozták (3-1 ábra), Git verziókezelést használva.



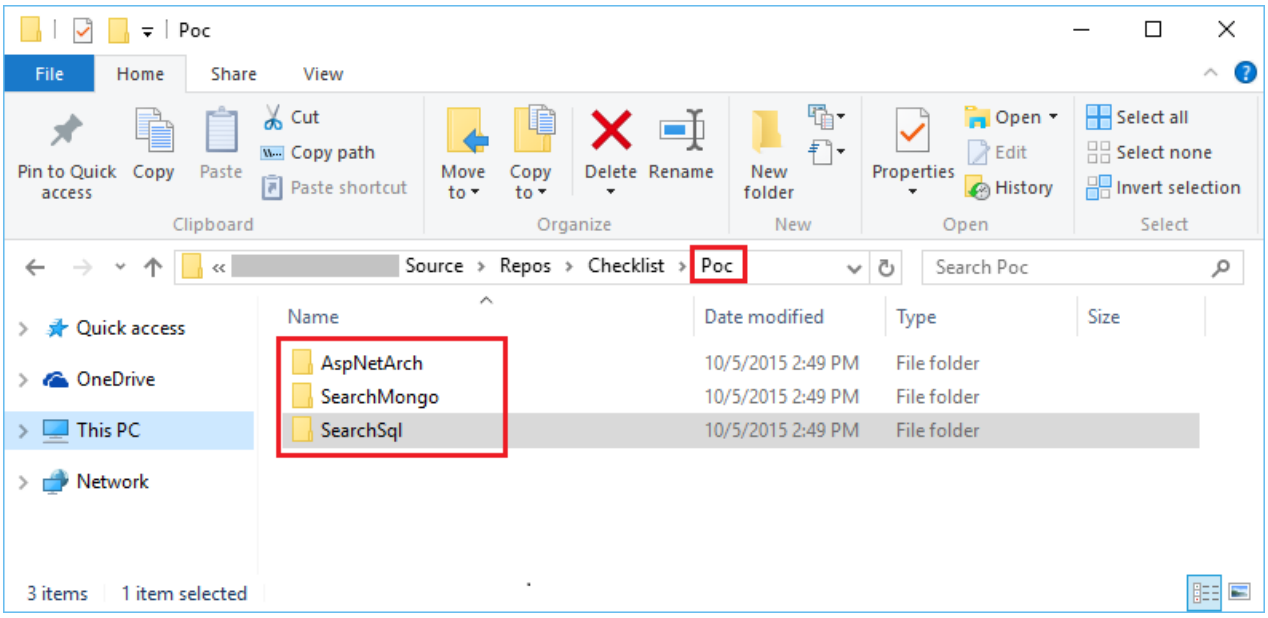
3-1 ábra: Git verziótár létrehozása a projekt kialakításakor

A fejlesztők a Visual Studio 2015 elindítása után egyszerűen hozzá tudtak kapcsolódni a verziótárhoz a Team Explorer segítségével, a **`git clone`** parancsnak megfelelő funkció használatával (3-2 ábra).



3-2 ábra: A Git tár lokális másolata (git clone)

A csapat számára fontos, hogy a fejlesztés során elkészített deszkamodellek is a forráskód részét jelentsék, ezért olyan mappaszerkezetet hoznak létre, ahol a fent említett deszkamodell projektek is helyet kapnak (3-3- ábra).



3-3 ábra: A deszkamodellek helye a forrásfában

A deszkamodellek elkészítése

József, Anita és László ugyan önállóan készítik el a deszkamodelleket, de ezek létrehozása közben folyamatosan megbeszélik a tapasztalatokat, segítik egymás munkáját. József az, amelyikük legtöbbet foglalkozott korábban az új ASP.NET-tel, ezért ő mutatja meg a projekt létrehozásának első lépéseit. Anita jól ismeri a MongoDb-t, László az SQL Servert, így nem véletlen, hogy ennek megfelelően osztották szét a feladatokat.

Anitának és Lászlónak egyaránt szüksége van egy olyan megoldásra, amely előállít 100 000 generált ellenőrző listát, kettejük egyeztetése alapján ezt László készíti el. Ezalatt Anita a kereséshez szükséges alkalmazások vázát készíti el, amelybe a későbbiek során már csak az adatbázis-specifikus programrészeket kell beilleszteni.

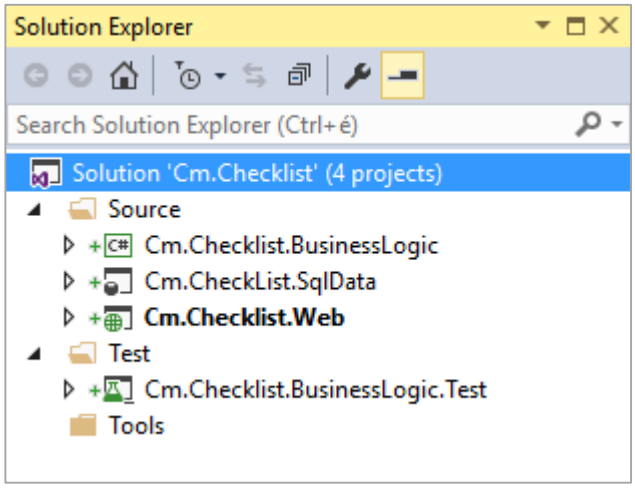
A deszkamodellek elemzése, kiértékelése

Mindhárom fejlesztő szeretné a rendelkezésre álló három napot a leghatékonyabban felhasználni. Egymás között úgy egyeztek meg, hogy az első napot a deszkamodell kialakításával töltik, a másodikat pedig azok kiértékelésével, elemzésével, hogy a harmadik napot már a fejlesztőkörnyezet finomítására és a sztorik becslésére fordíthassák.

A második napra elkészülnek a deszkamodellek, a csapat egyenként megtekinti azokat. József archtektúrajavaslatát elfogadják, néhány komponens nevének megváltoztatását javasolják, hogy a nevek „beszédesebbek” legyenek. Anita és László eredményeit összehasonlítva azt tapasztalják, hogy a MongoDB változat esetén kb. 22%-kal gyorsabb a keresés, az SQL Server változat használata során rövidebb a kód egy a csapat által már korábban is használt kódkönyvtárnak köszönhetően. Mivel a csapat összességében jobban ismeri az SQL Servert, emellett a változat mellett döntenek, de közösen egy apró módosítást végrehajtank az architektúrán, amely könnyebbé teszi azt, hogy szükség esetén akár MongoDB-t is használhassanak a jövőben. László és József felvállalja, hogy igyekszik jobban megismerni a MongoDb-t, ehhez Anita jól használható könyvet is javasol.

A fejlesztőkörnyezet kialakítása

A tapasztalatok alapján összeállítják az ellenőrző lista alkalmazásához tartozó kódvázat, és ezt kiegészítik azokkal az elemekkel, amelyeket a csapat folyamatosan használ a minőségbiztosításhoz (3-4) ábra.



3-4 ábra: A megoldás kezdeti szerkezete

A csapat külön szerelvénybe teszi az adatbázis kezelését, az üzleti logikát tartalmazó elemeket és a webes elemeket (a frontendet és a webszolgáltatások homlokzatát egyaránt biztosító komponenst). Sejtik, hogy ennél kifinomultabb architektúrára lehet majd szükségük a fejlesztés előrehaladtával — a jelenlegi komponensek további szerelvényekre való darabolásával —, de egyelőre nem szeretnék jobban elbonyolítani az architektúrát a szükségesnél. Átgondolt névtérhasználattal lehetővé teszik azt, hogy ha erre a későbbiekben szükségük lesz, azt majd egyszerűen megtehessék.

A csapat fontosnak tartja a rendszer tesztelését, ezért az üzleti logika elemeinek ellenőrzéséhez már a fejlesztés kezdetén létrehozzák a tesztelést végző szerelvényt.

A folyamatos termékleszállítás szemléletének követéséhez a csapat saját eszközöket (elsősorban parancssoros segédprogramokat) fog létrehozni, ezeket az ellenőrző lista alkalmazás forrásától elkülönítve a Tools mappában fogják elhelyezni.

A kialakított indulókészletet a Git verziótárba helyezik.

4. Termékkibocsátások tervezése

A csapat az eddig összegyűjtött ismeretei segítségével — beleértve a Tiborral egyeztetett sztorikat és a deszkamodellek megalkotásának eredményeit, hozzálát az eddig összegyűjtött sztorik becsléséhez azért, hogy azok segítségével kialakíthassák a következő néhány fejlesztési szakasz tartalmát.

Becslések sztoripontokkal

A legtöbb fejlesztő tudja, hogy az egyik legnehezebb dolog a becslés, vagyis megmondani, hogy egy adott feladat végrehajtása mennyi időt fog igénybe venni.

Még ha pontosan ismerjük is a megvalósítandó feladatot, annak megvalósítási ideje akkor is sok dologtól függhet. Milyen technológiát vagy eszközt használunk hozzá? Ki lesz az a fejlesztő, aki megvalósítja? Ha olyan, aki tapasztalt az adott témakörben, nyilván sokkal gyorsabban készül el, mintha egy kezdő fejlesztő lát hozzá.

A szoftver technológiával foglalkozó kutatások kimutatták, hogy a fejlesztők a kisméretű, jól megfogható feladatokat viszonylag pontosan, kb. 20%-os hibahatáron belül meg tudják becsülni. Ahogyan azonban a feladatok mérete nő, a fejlesztők egyre nagyobbat tévednek a becslések során – általában alábecsülik a ráfordításokat –, akár tízszeresen is.

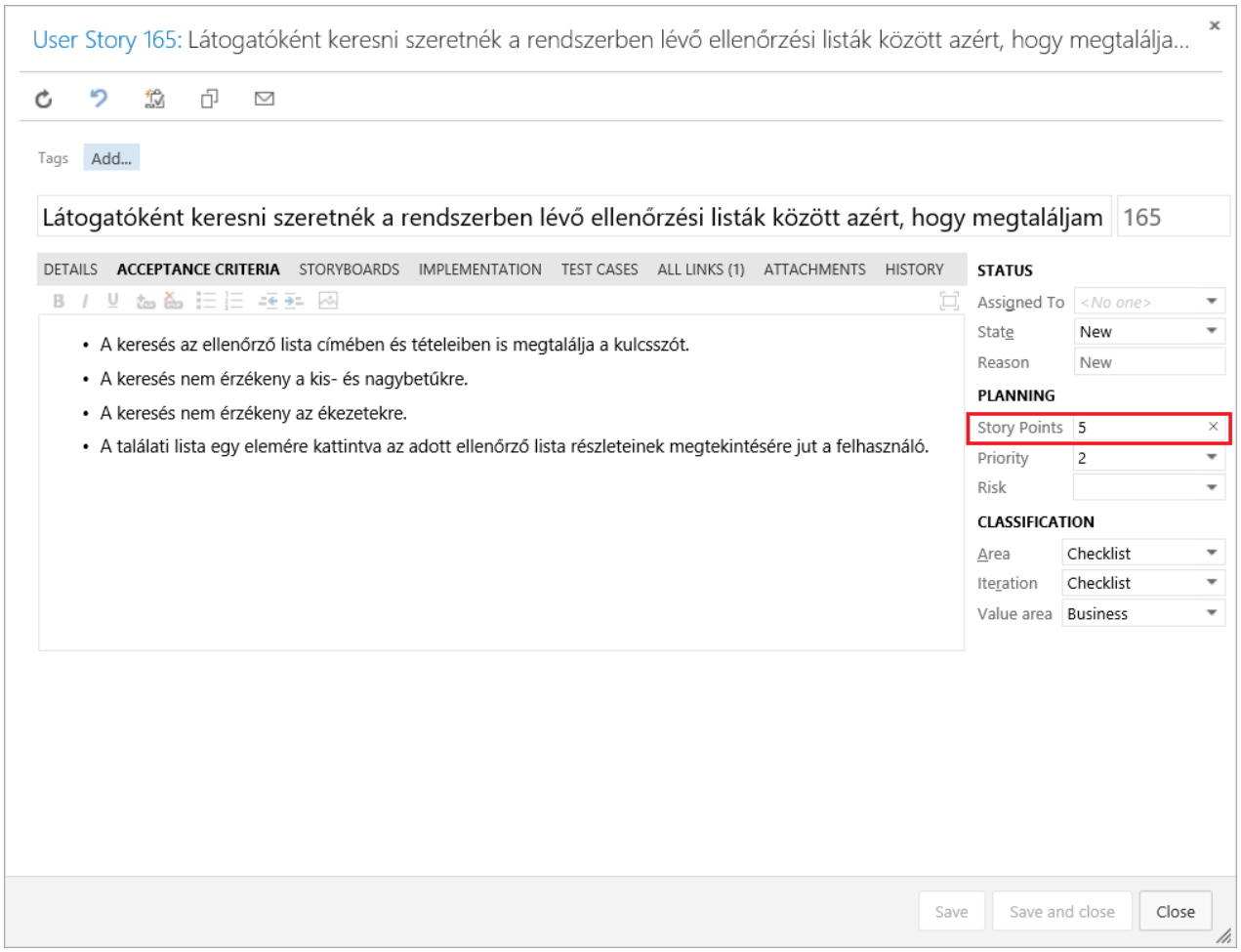
A csapat ezt a problémát úgy kerüli el, hogy egyes feladatok megvalósítását nem időben méri, hanem sztoripontokban: például azt mondja, hogy az egyik feladat megvalósítását 5 pontra, egy másikét 13 pontra becsüli. Ez első hallásra nagyon furcsának tűnik, de ha megértjük a mögötte lévő igencsak gyakorlatias megközelítésmódot, akkor mindez érthetővé válik.

A sztoripontos becslések során a pontszámok segítségével egymáshoz viszonyíthatók a feladatok. Egy 20 pontos feladat azt jelenti, hogy a csapat azt gondolja róla, az körülbelül négyszer annyi ráfordítást igényel, mint egy 5 pontos feladat, és nagyjából másfélszer annyit, mint egy 13 pontos. Ez a viszonyításon alapuló becslés sokat segíthet, mert általában könnyebben és pontosabban tudjuk két feladat egymáshoz viszonyított méretét megbecsülni, mint abszolút méretüket.

Az ellenőrző lista sztorijainak becslése

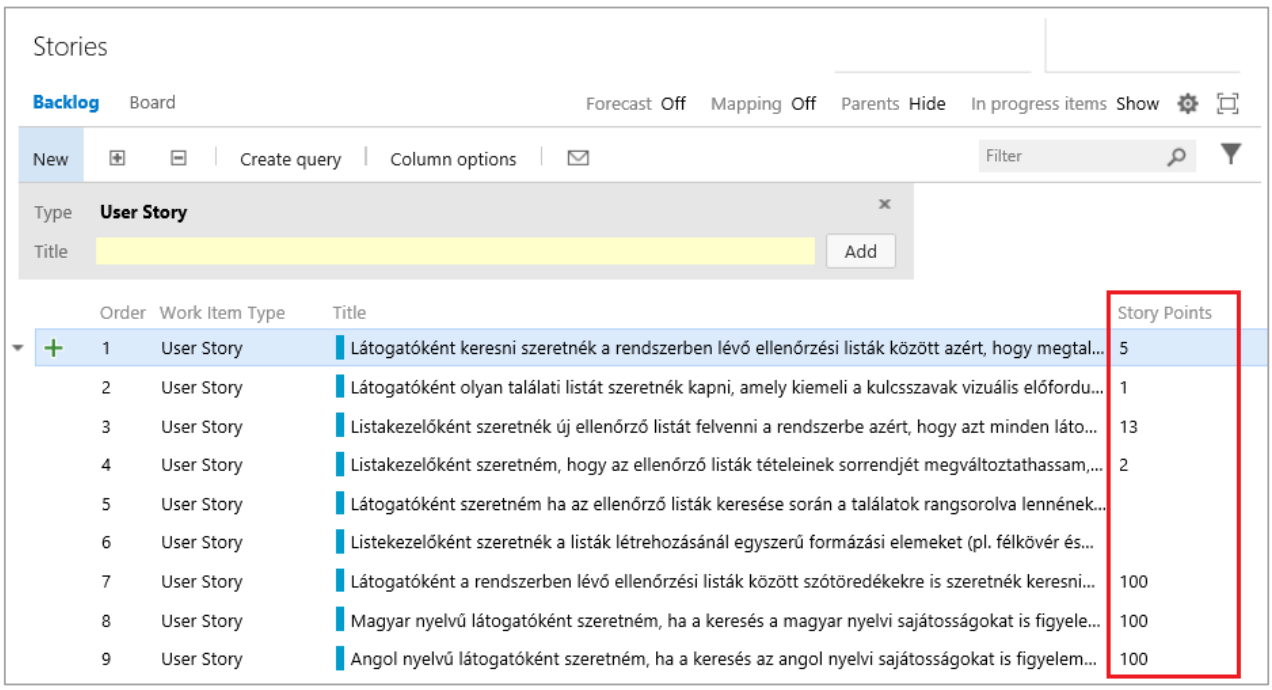
Az ellenőrző lista alkalmazáson dolgozó kis csapat már elegendő információval rendelkezik ahhoz, hogy a product backlogban lévő sztorik méretét — sztoripontokban — megbecsüljék.

Az ellenőrző lista projekt létrehozásakor a csapat a Scrum sablont választotta ki, így a product backlogban lévő bejegyzésekhez sztoripontokat rögzíthet, amint azt a 4-1 ábra mutatja.



4-1 ábra: Sztoripont rögzítése a product backlog elemeihez

A csapat a deszkamodelleknek megfelelően közösen — korábbi feladataikhoz viszonyítva — a kereséshez tartozó sztori méretét 5 pontra becsülte. Az összes sztori végig nézése után a 4-2 ábrán látható pontokat rendelték azokhoz.



4-2 ábra: Az ellenőrző lista sztorijaihoz tartozó pontszámok

Ebből a listából látszik, hogy a 3. sztori (új lista létrehozása) a 13 pontjával közel háromszor akkora feladat, mint az 1. sztoriban leírt keresés megvalósítása, amely csak 5 pont. A lista végén szereplő sztorikat a csapat egységesen 100 pontosnak látta, amely azt mutatja, hogy azok a jelenlegi formájukban igencsak nagyok — ezeket valószínűleg jelentősen egyszerűsíteni kell, vagy valamilyen módon a bennük lévő kockázatokat csökkenteni. Az 5. és 6. sztorira a csapat nem adott becslést, mert a sztori megválaszolatlan kérdéseket tartalmazott. Ezeket közösen megbeszélték Tiborral, aki termékgazdaként igyekszik válaszokat találni azokra, hogy a csapat egy közösen érthető, világos feladattal álljon szemben.

A csapat sebessége

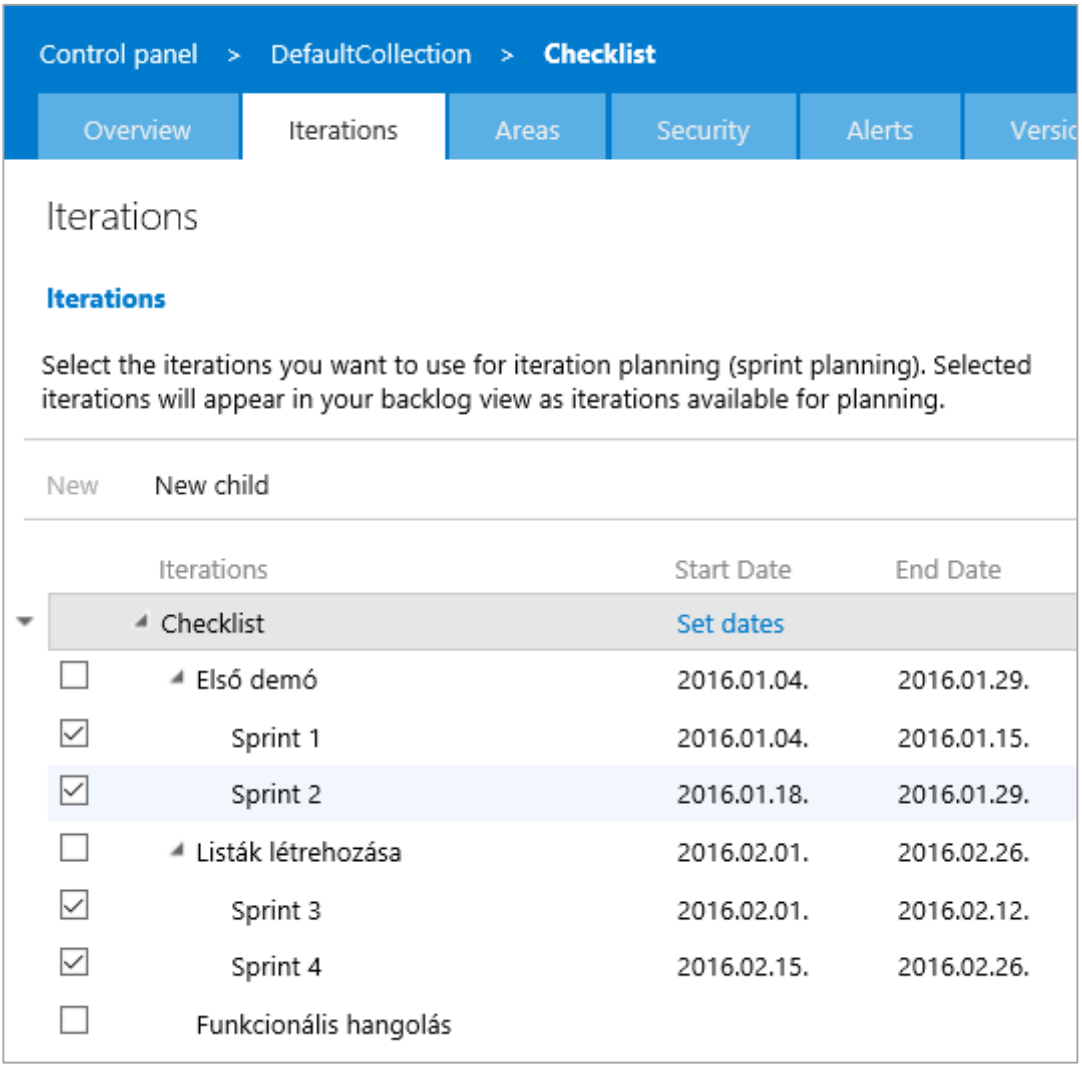
Egy jól összeszokott csapat — amelyik már hosszabb ideje használja a sztoripontos mérést — általában egy adott környezetben jól meghatározott *sebesség*gel rendelkezik. Ez azt jelenti, hogy egy fix — és a csapat által rendszeresen alkalmazott — időkeretben hány sztoripontnak megfelelő tevékenységet tud végrehajtani. Természetesen ez nem az első pillanatban alakul ki.

Az ellenőrző lista alkalmazáson dolgozó kis csapat már többször dolgozott együtt, és ismerik saját sebességüket, amely 20 sztoripont egy kéthetes ciklusban.

Iterációk és termékkibocsátások

A csapatnak két iteráció — két egymást követő kéthetes ciklus — áll rendelkezésére ahhoz, hogy a demóra felkészüljön. A demó szerves részének tekinti a csapat, hogy az elkészült alkalmazást a demó környezetbe telepítse — lehetőleg automatikusan —, hogy ezzel is ellenőrizze a kódépítési folyamat automatizáltságát.

Közösen leülnek, és átgondolják, hogyan fogják a terméket kibocsátani. Úgy döntenek, hogy a kezdeti időszakban minden második ciklus végén ki fognak bocsátani egy publikus, vagyis a felhasználók által éles termékként használható változatot. A csapat a 4-3 ábrán látható iterációkat és termékkibocsátási pontokat tervezi — ezeket a projektportálon állítják össze.

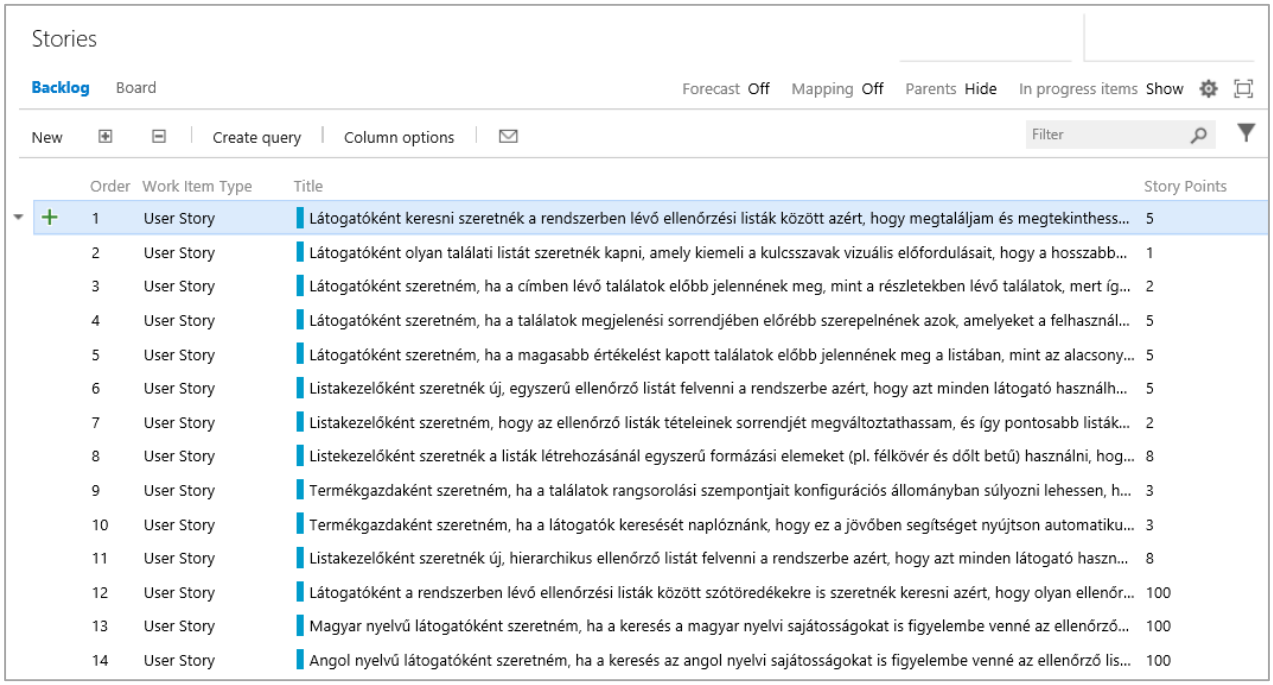


4-3 ábra: A tervezett iterációk

A csapat az iterációk tervezése során néhány fontos dologgal szembesül. A demóig két fejlesztési ciklus (sprint) van, a csapat sebessége 20 sztoripont, vagyis a demóig 40 sztoripontnak megfelelő feladatot tudnának elvégezni. A 4-2 ábra alapján azonban csak 21 pontnyi feladatuk van kifejtve, illetve a lista végén néhány túlsúlyos feladat szerepel. A 21 pontnyi kifejtett feladatból 13 pont az új ellenőrző listák létrehozására vonatkozik, pedig a csapat és Tibor szerint is a demóra sokkal fontosabb volna az, hogy a keresés legyen jól bemutatható.

Iterációk tartalmának hangolása

A csapat leül, és átalakítja a jelenlegi product backlogot, amint az a 4-4 ábrán látható.



Order	Work Item Type	Title	Story Points
1	User Story	Látogatóként keresni szeretnék a rendszerben lévő ellenőrzési listák között azért, hogy megtaláljam és megtekinthess...	5
2	User Story	Látogatóként olyan találati listát szeretnék kapni, amely kiemeli a kulcsszavak vizuális előfordulásait, hogy a hosszabb...	1
3	User Story	Látogatóként szeretném, ha a címben lévő találatok előbb jelennének meg, mint a részletekben lévő találatok, mert íg...	2
4	User Story	Látogatóként szeretném, ha a találatok megjelenési sorrendjében előrébb szerepelnének azok, amelyeket a felhasznál...	5
5	User Story	Látogatóként szeretném, ha a magasabb értékelést kapott találatok előbb jelennének meg a listában, mint az alacsony...	5
6	User Story	Listakezelőként szeretnék új, egyszerű ellenőrző listát felvenni a rendszerbe azért, hogy azt minden látogató használh...	5
7	User Story	Listakezelőként szeretném, hogy az ellenőrző listák tételeinek sorrendjét megváltoztathassam, és így pontosabb listák...	2
8	User Story	Listekezelőként szeretnék a listák létrehozásánál egyszerű formázási elemeket (pl. félkövér és dőlt betű) használni, hog...	8
9	User Story	Termékgazdaként szeretném, ha a találatok rangsorolási szempontjait konfigurációs állományban súlyozni lehessen, h...	3
10	User Story	Termékgazdaként szeretném, ha a látogatók keresését naplóznánk, hogy ez a jövőben segítséget nyújtson automatiku...	3
11	User Story	Listakezelőként szeretnék új, hierarchikus ellenőrző listát felvenni a rendszerbe azért, hogy azt minden látogató haszn...	8
12	User Story	Látogatóként a rendszerben lévő ellenőrzési listák között szótöredékekre is szeretnék keresni azért, hogy olyan ellenőr...	100
13	User Story	Magyar nyelvű látogatóként szeretném, ha a keresés a magyar nyelvi sajátosságokat is figyelembe venné az ellenőrző...	100
14	User Story	Angol nyelvű látogatóként szeretném, ha a keresés az angol nyelvi sajátosságokat is figyelembe venné az ellenőrző lis...	100

4-4 ábra: Az átalakított product backlog

Az átalakítás lényege, hogy a nyitott kérdéseket tartalmazó sztorikban lévő kérdéseket megválaszolják, és önálló sztorikba emelik ki azokat a részeket, amelyek zárt, egész termékképességeket alkotnak. Hasonló módon, szétbontják az ellenőrző listák létrehozásához tartozó sztorikat is kezelhetőbb méretűre.

A csapat becslést is végez, így sztoripontok kerülnek az egyeztett sztorikhoz.

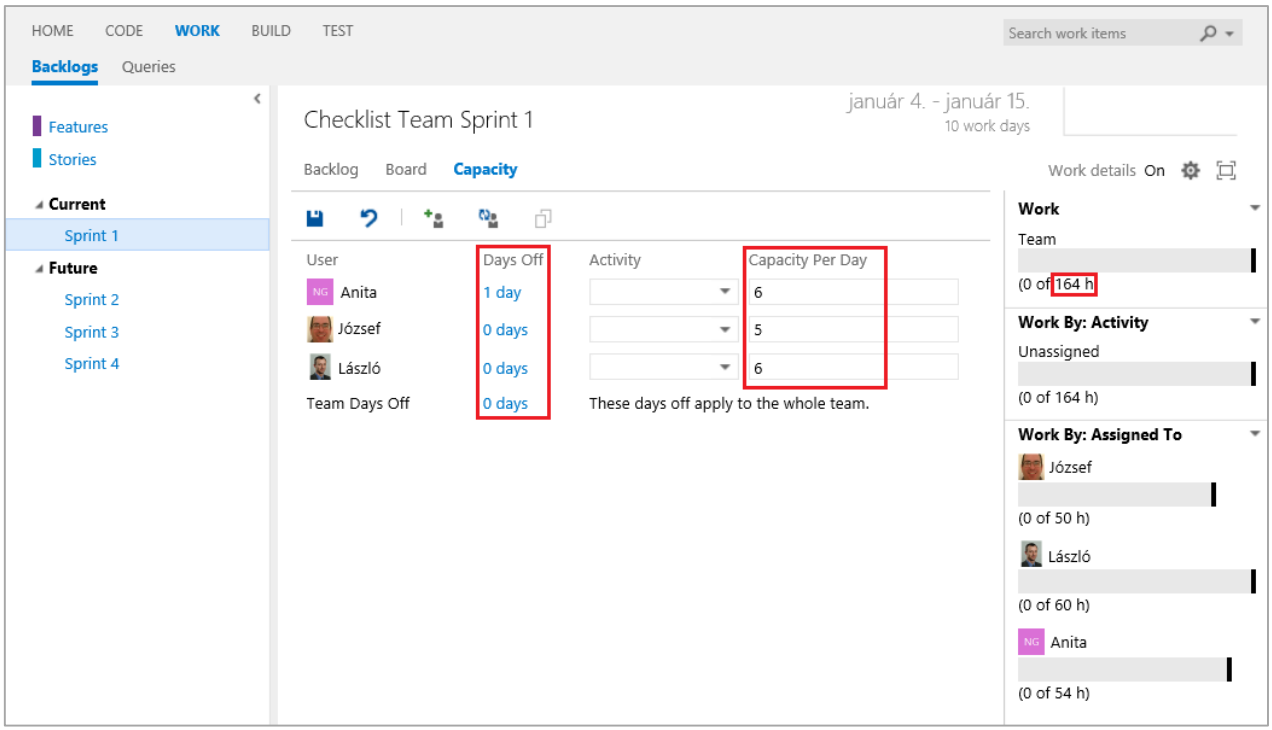
5. Fejlesztési ciklusok és feladatkezelés

Már elegendő információ áll a csapat rendelkezésére ahhoz, hogy az első két fejlesztési ciklust az üzleti értékeknek megfelelően feltöltsék elkészítendő sztorikkal. Tiborral együtt hozzálátnak ahhoz, hogy megtervezzék az első fejlesztési ciklust (sprintet).

A tervezési lépés során a sztorikat feladatokra bontják, és ezeket a feladatokat már munkaórában becsülik meg.

Kapacitásbecslés

Mielőtt a csapat a részletes tervezésbe belekezdene, tudniuk kell, hogy mennyi emberi erőforrással gazdálkodhatnak, vagyis azt, hogy mekkora a csapat kapacitása. Ennek megbecsléséhez a projekt portálja nyújt segítséget (5-1 ábra). A kapacitásbecsléshez tudni kell, hogy egy adott csapattag a munkanapjából mennyi időt tud a termékfejlesztésre fordítani. Mindhárom csapattag a munkaidejének egy részében más témájú megbeszéléseken szokott részt venni, illetve a munkahelyi levelezésük ellenőrzésére is időt fordítanak. Ezért Anita és László napi 6 munkaórát tud erre a projektre fordítani. Józsefet alkalomszerűen segítségül szokták kérni más csapatok, heti 3-5 órában, ezért az ő esetében napi 5 órával számolnak.



5-1: A csapat kapacitása

Anita a kéthetes sprint alatt egy napot egy képzésen tölt, távol a projekttől. Ezeket az adatokat a projektportálon megadva kiderül, hogy a csapat kapacitása 164 munkaóra a sprint során. A tervezésnél ezt figyelembe veszik.

A sprint backlog

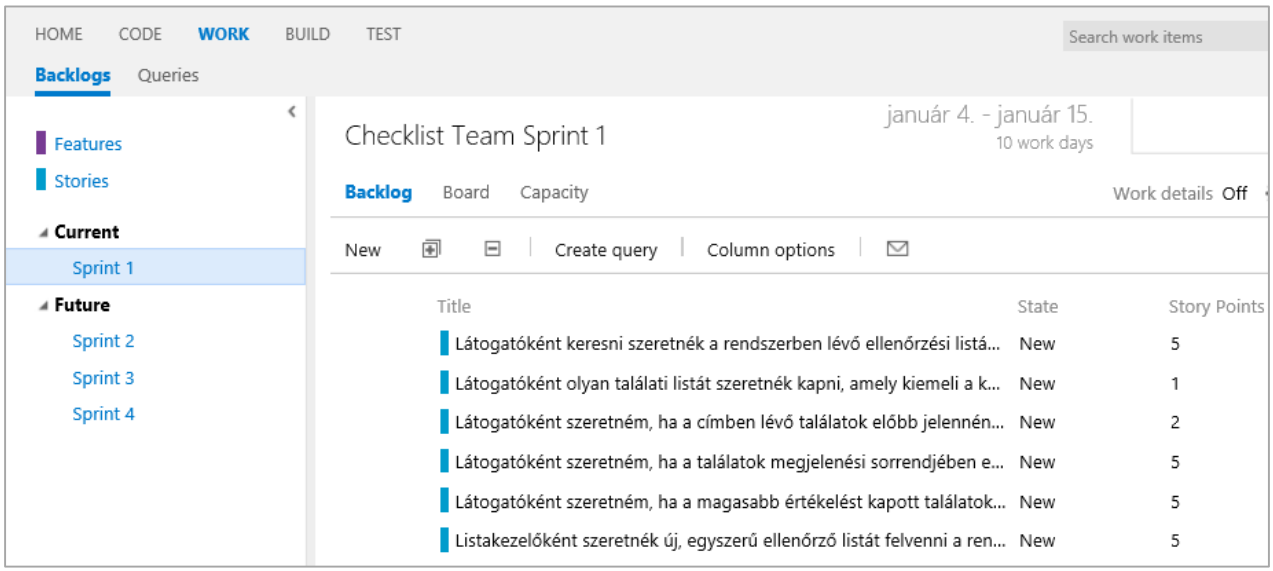
A sprint tervezése során a fejlesztőcsapat a product backlogból a sprint backlogba helyezi át azokat a sztorikat és egyéb teendőket, amelyeket az adott sprint során meg kíván valósítani. Ezek a bejegyzések értelemszerűen akkor helyezhetők át, ha nem tartalmaznak olyan nyitott kérdéseket, kockázati elemeket, amelyek megakadályozzák, hogy a csapat azokat megvalósíthassa. Mindennek olyan mértékben tisztának kell lennie, hogy a csapat nyugodt lelkiismerettel kezdhessen hozzá a tervezéshez és a fejlesztéshez.

Ha az adott sztori tervezése során — még a sprint tervezése alatt — olyan probléma merül fel, amely a megvalósítást megakadályozná, akkor a csapat ezt megvitatja a termékgazdával.

Feladatok összegyűjtése

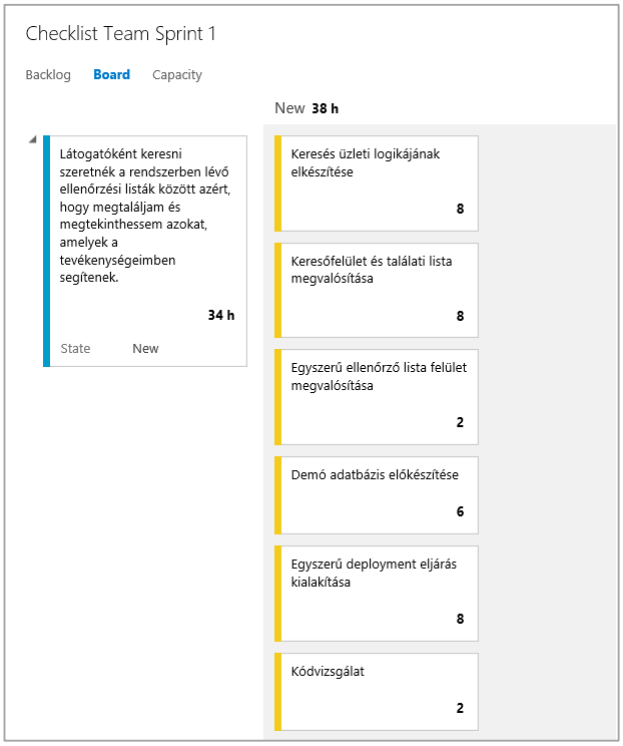
A csapat átlagos sebessége 20 sztoripont. A product backlog első öt eleme összesen 18, első hat eleme összesen 23 sztoripontot tesz ki, vagyis a csapat várhatóan öt, illetve hat sztorit vállal majd fel a kéthetes ciklus alatt. A sztorik átemelésekor feladatokat hoznak létre, és azokhoz már munkaórákat rendelnek, amely értelemszerűen csökkenti a maradék kapacitásukat — vagyis már nem sztoripontokat használnak.

Tibor az első hat elemet a sprint backlogjához rendeli (5-2 ábra), és átadja azokat a csapatnak azért, hogy megtervezzék a feladatokat.



5-2 ábra: A sprint backlog a tervezés elején

A csapat sztorinként végighalad a sprint backlog elemein, és azokat egyenként feladatokra bontják. Az 5-3 ábrán látszik az, hogyan is bontotta fel a csapat a legelső sztorit feladatokra. Minden feladat a „New” állapotban várakozik, egészen addig, amíg azzal valamelyik csapattag ténylegesen foglalkozni nem kezd.



5-3 ábra: Egy sztori felosztása feladatokra, időbecsléssel

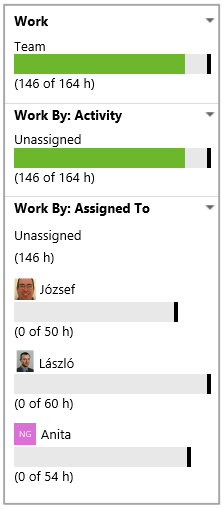
Minden feladat jobb alsó sarkában ott szerepel az annak megvalósításához kapcsolódó becsült munkaóra. Figyeld meg, a csapat nem hozott létre olyan feladatot, amely az egyes fejlesztési tevékenységek teszteléséhez kell! Ennek oka nagyon egyszerű: a fejlesztőcsapat számára teljesen egyértelmű, hogy egy fejlesztési feladat csak a hozzá tartozó tesztekkel együtt van kész.

Azt is láthatod, hogy ezzel ellentmondásnak lévőnek tűnő módon viszont létrehoztak egy feladatot a kódvizsgálatra. Ennek oka az, hogy a csapat már olyan minőségben dolgozik, hogy nem minden sztori esetében csinálnak önálló kódvizsgálatot: a közös munka során már maguktól felhívják társaik figyelmét azokra az apróbb kódminőségi problémákra, amit találnak, és azokat azonnal ki is javítják.

Ez a sztori azonban még újszerű elemeket tartalmazhat, ezért a csapat úgy dönt, hogy közösen is átvizsgálják a kódot — ezért ennek megfelelően felvesznek erre egy feladatot.

A csapat — a demó forgatókönyvnek köszönhetően — folyamatosan azt tartja szem előtt, hogy demóznia kell a sztorit, illetve automatizálnia kell a demó (és később az egyéb) környezetekbe való telepítést, ezért létrehozzák az ennek megfelelő tevékenységeket is.

Az összes sztori részletes tervezése után a csapat a 164 óra kapacitásából 146 órát tervez felhasználni, amint azt az 5-4 ábra mutatja.



5-4 ábra: Feladatokhoz rendelt kapacitások a sprint tervezése után

Célok kijelölése, vállalás

Anita, József és László meggyőződtek arról, hogy a sprintre tervezett feladatok végrehajthatók, és örülnek, hogy a sprintbe 23 sztoripontnak megfelelő feladatot tudtak bevenni. Tiborral egyeztetve a csapat vállalást tesz arra, hogy a kiválasztott sztorikat sikeresen, az elvárt minőségben készre jelentik az iteráció végén — és azt demonstrálják is — első körben csak Tibornak és néhány belső érintettnek (marketing vezető, üzemeltetés képviselője). Megfogalmazzák a sprint célját is, amelyet az „egyszerű, de látványos keresés működik" mondatban fogalmazzák meg.

Közösen elindítják a sprintet, vagyis a következő hétfőtől a csapat hozzákezd az ellenőrző lista alkalmazás megvalósításához.

Napi egyeztetés

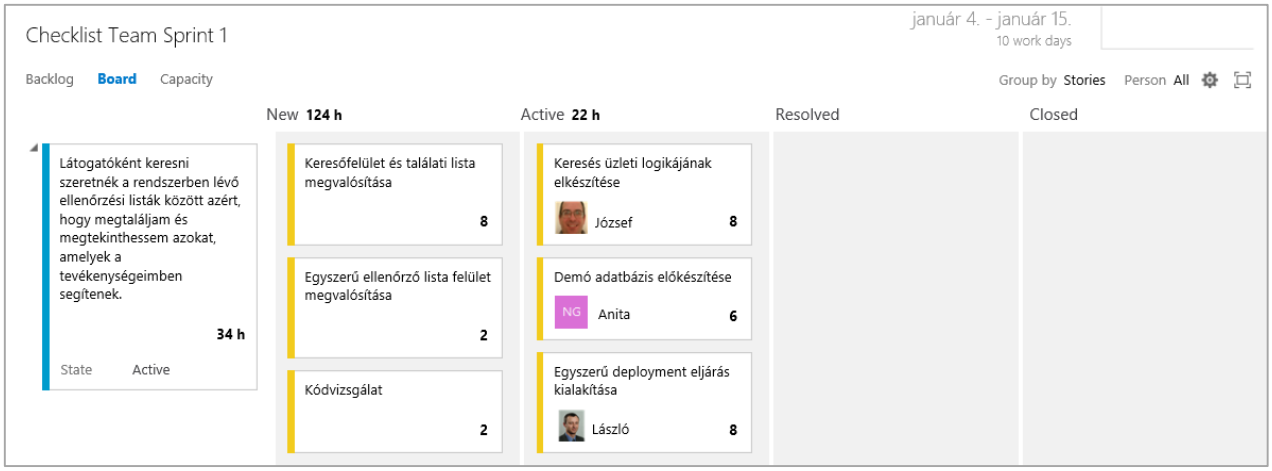
A csapat minden nap egy közös egyeztetést tart, amely nagyon rövid (legfeljebb 10 perc), és az alábbi kérdésekre fókuszál:

Mit csináltam az előző napi megbeszélés óta, ami hozzájárult ahhoz, hogy a csapat elérje a sprint célját?

Mit fogok tenni a következő napi megbeszélésig, ami tovább viszi 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 témákról a napi megbeszélésen nem esik szó. Ha a csapat úgy dönt, hogy egy kérdést részletesen meg kell vitatni, akkor azt más keretek között teszik meg.
Az egyeztetés során a csapattagok felvállalják azokat a feladatokat, amelyekkel a következő egyeztetésig foglalkozni szeretnének. Az 5-5 ábra azt mutatja meg a portál feladatkövetést végző felületén.



5-5 ábra: Feladatok aktiválása

Csapatmunka

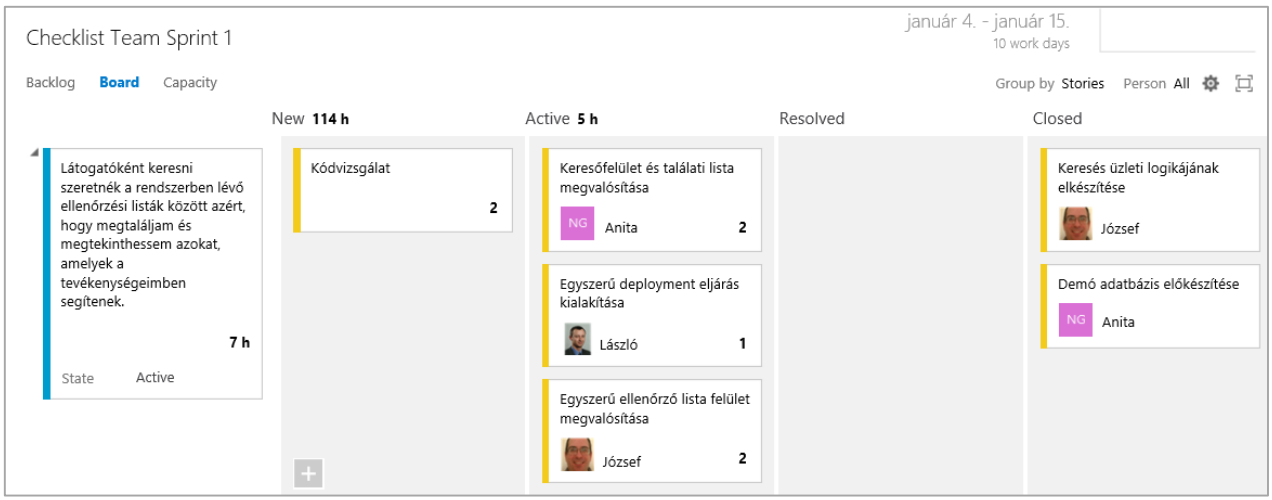
A „hagyományos” fejlesztési projekteken általában minden fejlesztőnek van egy-egye olyan szakterülete, amelyen nagyobb tapasztalata van, mint a többieknek, illetve egy adott projektcsapaton belül hatékonyabb a többieknél. Ez ebben a csapatban is így van. József esetében ez az adatbázis-programozás, Lászlónál az üzleti műveletek megvalósítása, Anita pedig legjobban a felhasználói felületek előállításához ért.

A csapat számára vonzó megoldásnak tűnik, hogy mindenki a sprint feladatai közül magához ragadja azokat, amelyek a saját specialitásának megfelelnek. Ebben az esetben József magára venné az összes adatbázis feladatot, Anita pedig a felhasználói felületek készítését.

A csapat azonban jól tudja, hogy ez a megközelítésmód nem igazán hatékony és eredményes, mert a sprint végére tolja a feladatok során keletkező munkadarabok integrálásának problémáit, késlelteti a tesztelést, és így veszélyes.

A hatékony együttműködésnek az a kulcsa, hogy a csapat, ameddig csak lehetséges, egy sztori megvalósítására koncentrál, és mindenki azon dolgozik. Több sztorihoz egyidőben csak akkor nyúlnak, ha egy sztorihoz már nem fér optimálisan hozzá a csapat.

A napi egyeztetéseken a csapat folyamatosan megbecsüli, hogy a még el nem készült feladatok befejezése várhatóan mennyi időt vesz igénybe. Az 5-6 ábrán ezt láthatjuk.



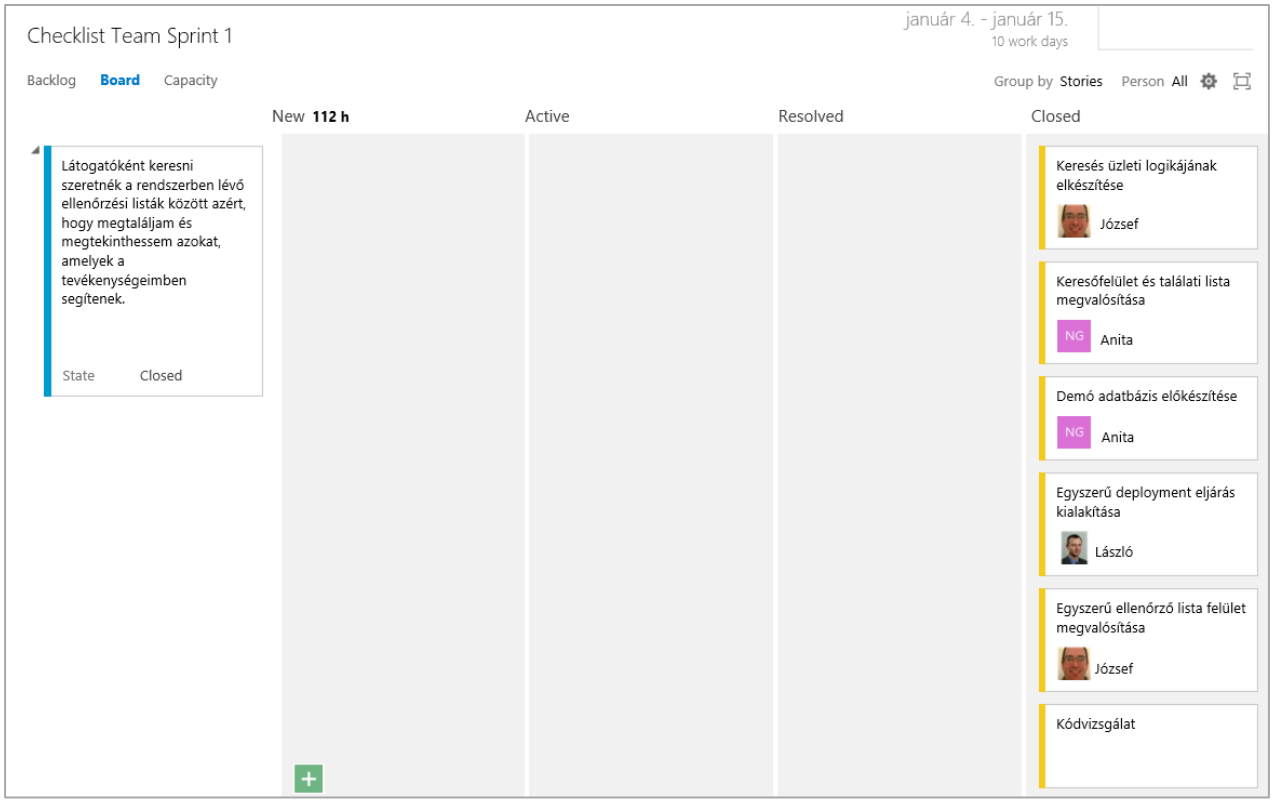
5-6 ábra: A feladatok várható befejezésének követése

Készre jelentés

A csapat egy sztorival elvileg akkor készül el, amikor befejezi az összes ahhoz tartozó feladatot. Az agilis csapatok azonban olyan elfogadási kritériumokkal is rendelkezhetnek, amelyek egységesen, minden sztorira vonatkoznak, anélkül, hogy azokat explicit módon felsoroltuk volna. Ezeket a csapat egy listán vezeti, amelyet az agilis keretrendszerek *definition of done* — a „kész" definíciója — néven ismernek.

A csapat ilyen kritériumoknak tekinti azt, hogy minden sztorihoz az elfogadási kritériumokat ellenőrz utomatikus teszteknek kell tartoznia. A csapat felveszi még erre a listára a dizájnnal és a felület viselkedésével kapcsolatos alapvető kritériumokat is.

Ha minden feladat a „kész" definíciója szerint is megvalósultnak tekinthető, a csapat késznek nyilváníthatja az adott sztorit (5-7 ábra).



5-7 ábra: A sztori elkészült

A sprint terjedelme akkor van kész, ha az összes sztori elkészült.

Abban az esetben, ha a csapatnak marad még szabad kapacitása, a termékgazdával, Tiborral egyeztetnek a teendőkről. Elképzelhető, hogy találnak egy olyan sztorit, amely még beemelhető a sprintbe, de az is lehet, hogy a maradék időt egyszerű deszkamodell megvalósítására használják, amely későbbiekben megvalósítandó sztorik lehetséges kockázatát csökkentik.

6. Technológiai sokszínűség

Az ellenőrző lista alkalmazás kis fejlesztőcsapata Az ASP.NET MVC 5 keretrendszert és az Angular JavaScript könyvtárat választotta ki technológiai bázisaként. A felhasználói felület leírásához a HTML és CSS nyelvek adják az alapot. A szerver oldali komponensek megvalósítására a C# programozási nyelvet, a frontend oldalon pedig a TypeScript nyelvet használják. Mivel SQL Server adatbázist használnak, ezért értelemszerűen a Transact-SQL nyelv segítségével írják le az adatkezelő műveleteket. A telepítéshez tartozó automatizmusokat PowerShell szkriptekkel, illetve a JavaScript alapú *gulp* komponensekkel állítja elő a csapat.

A fentieken kívül még számos apró nyelvi és technológiai elemet használ a csapat, beleértve az XML és JSON adatleírásokat, illetve néhány parancssori utasítást egybefogó batch fájlokat. Érdemes azt is megemlíteni, hogy az automatikus telepítéshez használt eszközök egy része Node.js alapú…

Ugye, milyen meglepő, hogy már egy kisebb termék fejlesztéséhez is ennyi programozási nyelv, könyvtár és komponens kerül be a csapat eszköztárába? Ezeket a csapat teljes egészében a Visual Studio IDE segítségével tudja használni.

Programozási nyelvek és technológiák

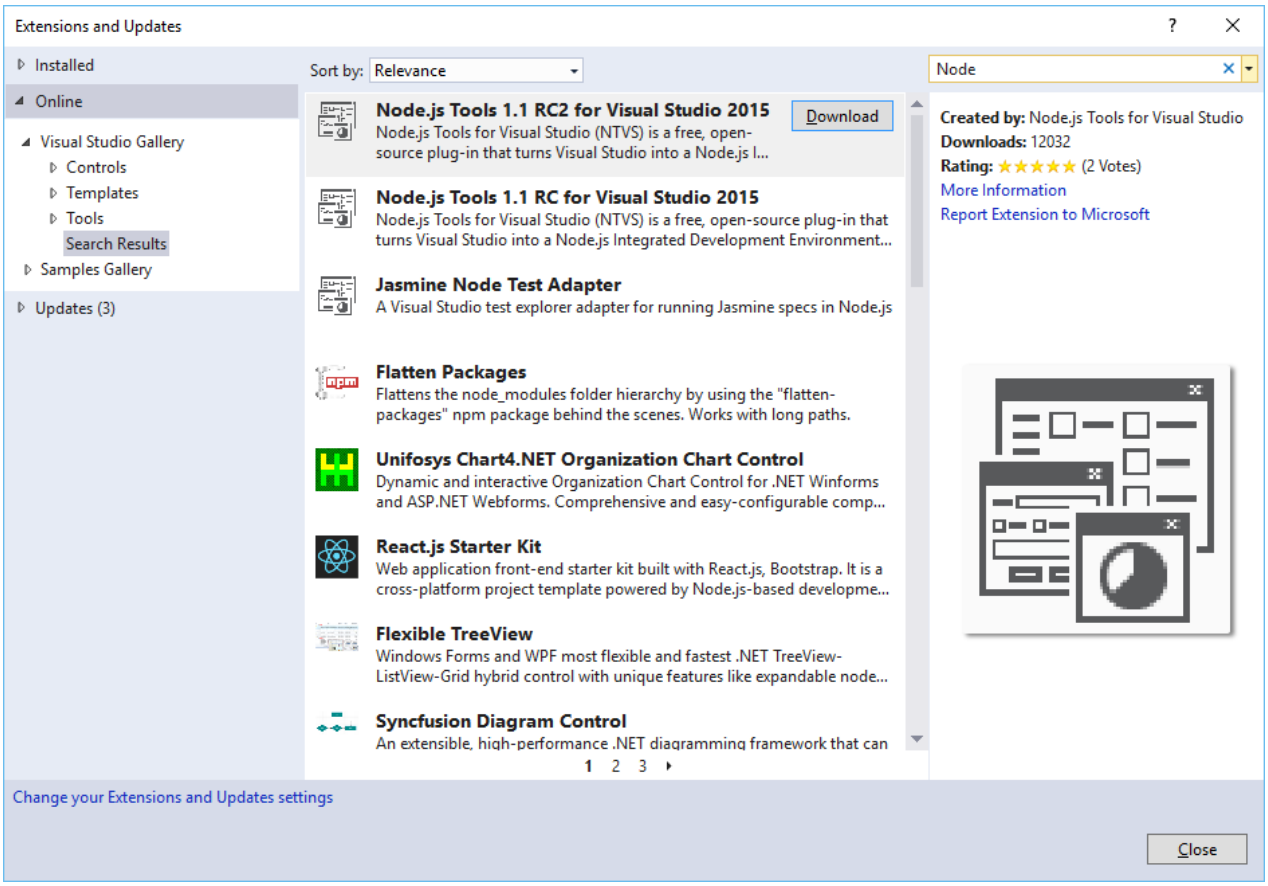
Az itt felsorolt programozási nyelvek, technológiák, keretrendszerek mellett a csapat az ellenőrző lista alkalmazás továbbfejlesztése során újabbakat fog bevonni eszköztárába. Amikor a webes megoldás irányából elmozdulnak a mobil fejlesztés irányába, akkor a Xamarin keresztplatformos megoldása vagy az Apache Cordoba között is választhatnak — és ehhez kis sem kell lépniük a Visual Studio kereteiből.

Bar a Visual Studio .NET-es változatának (Visual Studio .NET 2002) megjelenésekor a C#, Visual Basic, C++ és a HTML nyelvek voltak az IDE részei, ez a kör a jelenlegi — Visual Studio 2015 — változatig rendkívüli mértékben kibővült. A programozási nyelvek listáján ott találhatjuk az F#, Python, JavaScript, TypeScript nyelveket — azonnal a Visual Studio telepítése után. Bővítményként letölthető sok egyéb programozási nyelv a hozzájuk tartozó eszközkészlettel, mint például a PHP vagy a Node.js.

Az ellenőrző lista alkalmazás példája is mutatja, hogy ma már egy-egy fejlesztés kapcsán nem a programozási nyelvek, hanem az alkalmazás-fejlesztési modell irányából célszerű elindulni. A Visual Studio minden alapvető alkalmazásmodellt támogat, beleértve az asztali, mobil, webes, adatelemző, játék és felhőben futó alkalmazások fejlesztését is.

Visual Studio bővítmények

A Visual Studio használatával a fejlesztőcsapat nemcsak a termék telepítése során a számítógépre helyezett eszközöket kapja meg, hanem egyúttal egy olyan ökoszisztéma részévé is válik, amely segítségével rengeteg — és jelentős részben ingyenes — bővítményt telepíthet a Visual Studio kiegészítéseként. A 6-1 ábrán az IDE Extensions and Updates dialógusát láthatjuk, amint éppen a Node.js-hez kapcsolódó bővítményeket listázza.



6-1 ábra: Node.js-hez kapcsolódó Visual Studio bővítmények

Ha valamilyen feladathoz, technológiához, programozási nyelvhez kapcsolódóan eszközökre van szükséged, azokat valószínűleg megtalálod a már elkészített bővítmények között. Ha neked van egy jó elképzelésed, a megvalósított kiegészítőt megoszthatod a Visual Studio közösséggel.

Hasznos linkek

Az ellenőrző lista alkalmazás csapata hasznosnak találta az alábbi linkeket. Lehet, hogy te is találsz közöttük olyanokat, amelyek segítenek terméked fejlesztésében! [2]

Programozási nyelvek és eszközök

PHP Tools for Visual Studio

Node.js Tools for Visual Studio

Fejlesztési technológiák

Felhő alkalmazások fejlesztése

Általános alkalmazásfejlesztés (Windows Universal, mobil, keresztplatformos)

Webes alkalmazásfejlesztés (Node.js, ASP.NET, PHP, Java, Ruby)

Adatkezelés és elemzés (SQL Server, Big Data)

Játékfejlesztés

[2] A Hasznos Linkek szekció a hivatkozott oldalakra mutató linkek 2015.11.18-i állapotát tünteti fel.

6. Technológiai sokszínűség

Fejlesztői központok

Windows 10

Microsoft Azure

Visual Studio

Office 365

Xbox

SharePoint

Dynamics

Önálló tanulás

Microsoft Virtual Academy

Channel 9

Visual Studio bővítmények

Visual Studio Gallery

Visual Studio SDK

Nyílt forráskódú technológiák

ASP.NET a GitHub-on

ASP.NET vNext

További információ, támogatás

Microsoft API katalógus (Referencia dokumentációk)

Microsoft fejlesztői csoportok blogjai

Szakértők keresése

7. Minőségbiztosítás

A csapat az ellenőrző lista alkalmazás termékként való kezelésében kulcsát abban látja, hogy folyamatosan, jó minőségben adja át az elkészült sztorikat. Mivel viszonylag kis sztorikkal dolgoznak, ezért arra kell felkészülniük, hogy rendszeresen refaktorálniuk kell a kódot, vagyis annak belső struktúráját folyamatosan átkell alakítaniuk. A csapat számára kiemelt felelősség az alkalmazást folyamatosan olyan állapotban tartani, hogy az könnyen módosítható legyen és ne kelljen a fejlesztőcsapat tagjainak egymás kódjától félniük — képesek legyenek azokat egyszerűen megérteni, átalakítani.

Több eszközt is használnak ezeknek a céloknak az elérésére.

Automatikus tesztelés

A csapat alapvetőnek tekinti, hogy az ellenőrző lista alkalmazást folyamatosan tesztelniük kell, vagyis ellenőrizni, hogy az megfelel-e az elvárt funkcionalitásnak és a vele szemben támasztott elfogadási kritériumoknak. Ezeket a teszteket kézzel hatékonyan elvégezni szinte lehetetlen — mindenképpen nagyon költséges. A csapat a korábbi tapasztalatainak megfelelően automatizálja az ellenőrzéseket a fejlesztési folyamat során. Az automatizálás azt jelenti, hogy a tesztek önmaguk is futtatható munkadarabok, amelyek csak minimális emberi beavatkozást igényelnek (általában azt, hogy elindítsák őket), és lefutásuk után azonnal eldönthető, hogy a teljes tesztkészletből melyek voltak a sikeresek, és melyek utalnak arra, hogy valami még nem felel meg az elvárásoknak.

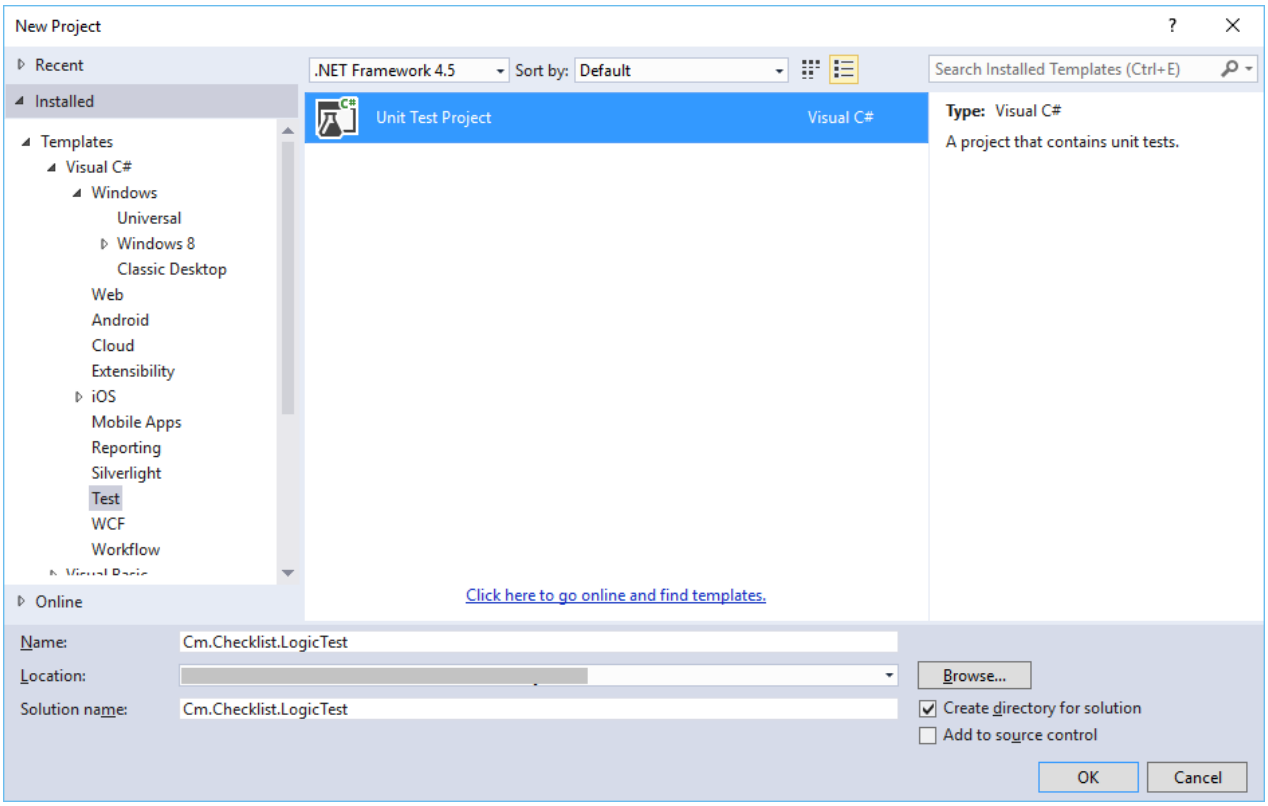
Anita, József és László minden egyes fejlesztési munkadarabhoz automatikus teszteket készít, amelyeket le is futtatnak, így ellenőrzik, hogy az adott feladat megfelel-e az elfogadási kritériumoknak. Úgy tekintik, hogy egy munkadarabnak szerves részét jelentik ezek a tesztek, így nem is hoznak létre önálló feladatokat ezek elkészítéséhez.

Az egyik sztori, amit megvalósítottak, így néz ki:

Látogatóként keresni szeretnék a rendszerben lévő ellenőrző listák között azért, hogy megtaláljam és megtekinthessem azokat, amelyek a tevékenységeimben segítenek.
Leírás: A kereséshez egyetlen szövegmezőbe beírt kulcsszót (akár több részből is állhat) használunk, a webes keresőkhöz hasonlóan. *Elfogadási kritériumok:* A keresés az ellenőrző lista címében és tételeiben is megtalálja a kulcsszót. A keresés nem érzékeny a kis- és nagybetűkre. A keresés nem érzékeny az ékezetekre. A találati lista egy elemére kattintva az adott ellenőrző lista részleteinek megtekintésére jut a felhasználó. *Demó forgatókönyv:* A sztori demójához egy demó/teszt adatbázist hozunk létre, azt előre feltöltjük egy szöveges teszt állományból. Az ellenőrző lista részletes nézete egyelőre csak egyszerűen megjeleníti a lista címét és elemeit. 1. Rákeresek a „hasra esés” kulcszóra: üres találati listát kapok. 2. Rákeresek az „árvíztűrő tükörfúrógép” kifejezésre: 2 találatot kapok, egyet a címben, egyet egy ellenőrző lista tételében. 3. Rákeresek az „arvizturo tukorfurogep” kifejezésre: 2 találatot kapok, egyet a címben, egyet egy ellenőrző lista tételében, pontosan úgy, mint az előző keresési lépésben. 4. Rákeresek a „ScRUm” kifejezésre: 4 találatot kapok, ezekben szerepelnek a „scrum”, „Scrum” és „SCRUM” szavak egyaránt. Megmutatok, hogy egy találatra kattintva az adott ellenőrző lista részletes nézetébe jutok.

A csapat a tesztelést annyire fontosnak tartja, hogy külön feladatot rendelt ehhez a sztorihoz, amely csak azzal foglalkozik, hogy a leírt demó forgatókönyveknek megfelelő tesztadatbázist hozzon létre. A teszteléshez és a fejlesztési ciklus végén lévő demóhoz is ezt a tesztadatbázist használják.

Bár a forgatókönyvben csak 4 ellenőrzendő helyzet szerepel, a csapat ennél jóval több automatikus tesztet hoz létre, amelyeket a Visual Studio 2015 Unit Test Project sablonjának segítségével (7-1 ábra) valósítanak meg.



7-1 ábra: Automatikus teszt létrehozása

A tesztesetek megvalósítása során a csapat olyan mintát követ, amely a teszteset előkészítését, végrehajtását és az elvégzendő ellenőrzéseket — ebben a sorrendben — tartalmazza. Alapvetően nem a felhasználói felület automatizálásán keresztül hajtják végre a teszteket, hanem már az architektúra kialakításakor figyelnek a tesztelhetőség megvalósítására.

A csapat 27 tesztesetet készít a keresés logikájának ellenőrzésére, amelyeknek beszédes neveket adnak:

#1 — **`SearchIsCaseInsensitive`**: értelemszerűen annak ellenőrzése, hogy a keresés a kis és nagy betűket nem különbözteti meg.

#2 — **`SearchInTitlesWorksAsExpected`**: annak ellenőrzése, hogy az ellenőrző lista címeiben a keresés helyesen működik

#3 — **`SearchInDetailsWorksAsExpected`**: annak ellenőrzése, hogy az ellenőrző lista részleteiben a keresés helyesen működik

#4 — **`SearchFailsWithNullRequest`**: a keresést megvalósító szolgáltatás hibának tekinti, ha *null* értéket adnak át neki a keresés paramétereit leíró adatként (ez egy belső, ún. white-box teszt).

...

#27 — **`SearchIsAccentInsensitive`**: annak ellenőrzése, hogy a keresés az ékezetes betűket úgy kezeli, mint az ékezet nélküli megfelelőiket.

A csapat a felhasználó felület logikájának — ez egy külön alkalmazásréteg! — ellenőrzésére is automatikus teszteket készít:

#1 — **`SearchIsNotTriggeredWithEmptyKey`**: a keresésnek csak akkor kell elindulnia, ha már beírt a felhasználó valamit a keresőfeltételek közé

#2 — **`SearchStartsWithDelayAfterTyping`**: ellenőrzi, hogy a keresés a keresőszöveg gépelése közben az utoljára leütött billentyű után egy rövid ideig várakozik, mielőtt a keresést elindítaná.

...

#14 — **`SearchCancelsPreviouslyStartedRequests`**: ellenőrzi, hogy a gépelés során a korábbi kereséseket a rendszer leállítja — mindig csak az utolsó keresést igyekszik befejezni — az erőforrások kímélése céljából.

Amire a csapat az automatikus tesztelést használja

Amellett, hogy a fejlesztés közben a készülő tesztek folyamatosan segítik a hibák megelőzését, illetve a felfedezett hibák gyors javítását, a csapat még két további fontos előnyhöz is jut:

Az elkészült automatikus tesztek alapvető feltételei a biztonságos refaktorálásnak, vagyis annak, hogy a termék (szoftver) belső szerkezét a csapat magabiztosan alakíthassa át, ne kelljen félnie, hogy egy ilyen tevékenység során újabb hibákat vezet be. Ezeket a teszteket a refaktorálás folyamata során a csapat gyakran használja, így azonnal észreveszik, ha a változtatások rontják a termék működését, megbízhatóságát.

Az automatikus tesztek egyúttal dokumentációként is használhatók a szoftver belső komponenseinek működésének, illetve azok homlokzatának leírásához.

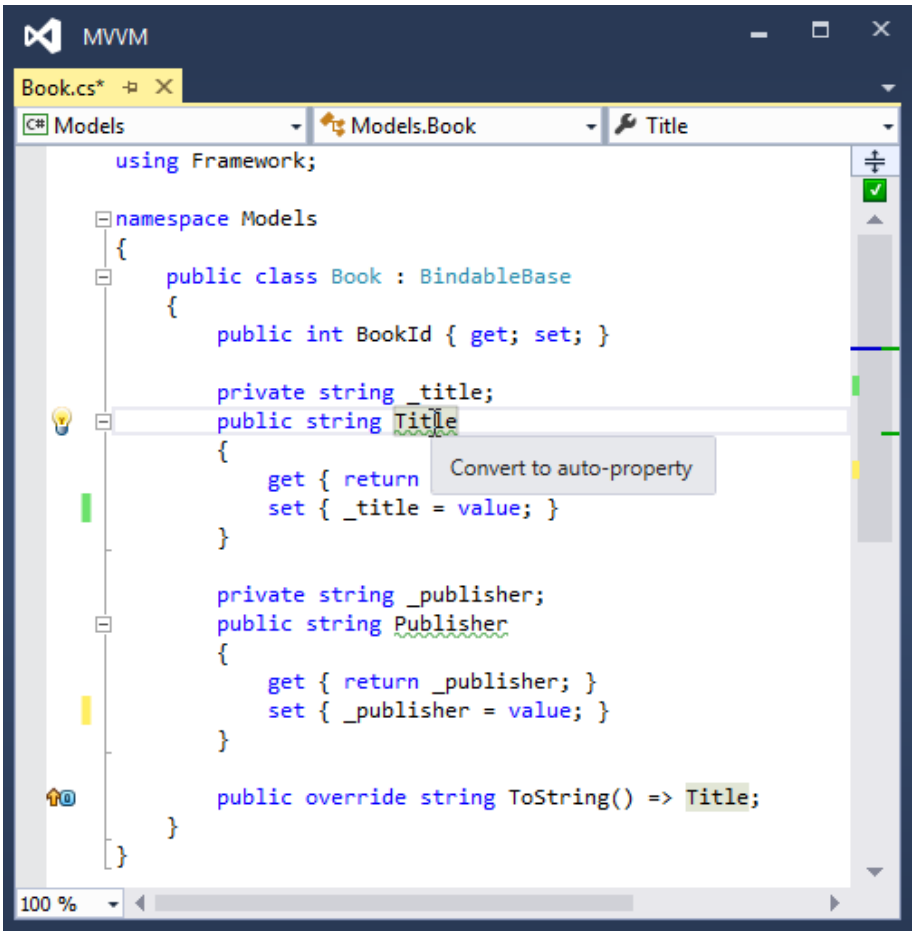
Refaktorálás

A Visual Studio 2015 a leggyakrabban használt refaktoring eljárások közül többet is támogat, ahogyan azt alábbi táblázat összefoglalja:

Funkció	**Leírás**
Rename	Azonosítók és kódszimbólumok (pl. típusok, műveletek, tulajdonságok, mezők, változók stb.) nevét változtatja meg
Encapsulate Field	Tulajdonságot épít egy meglévő mező köré
Remove Parameters	Műveletek, indexerek, delegáltak valamelyik paraméterét távolítja el
Reorder Parameters	Műveletek, indexerek, delegáltak paramétersorrendjét változtatja meg
Extract Interface	Egy meglévő osztály vagy struktúra tagjait kiemeli egy önálló interfészbe
Extract Method	Egy műveletben lévő kódtöredéket önálló műveletbe zárva kiemel

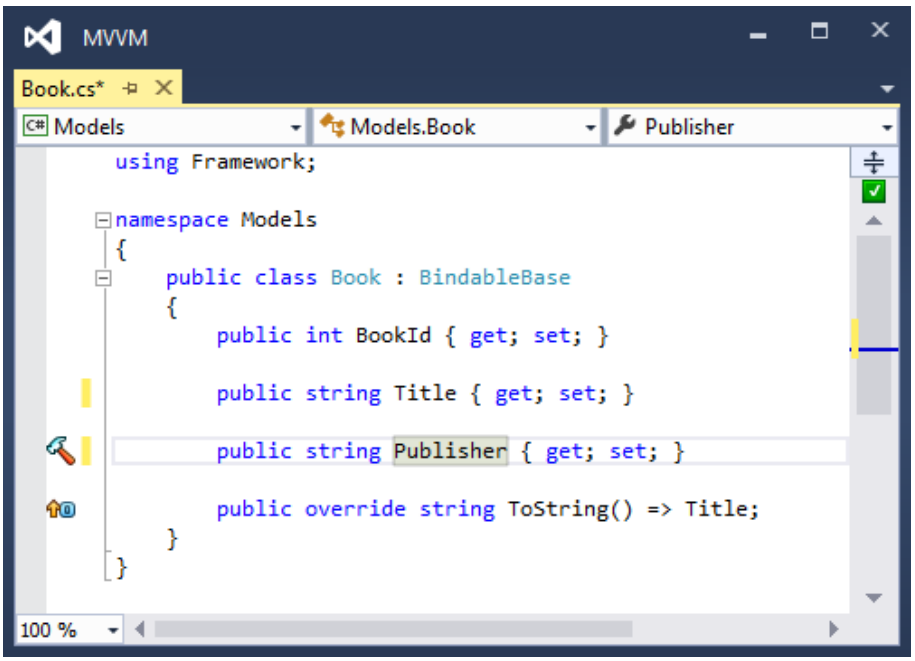
Értelemszerűen az összes refaktoring funkció gondoskodik arról, hogy az egyes kódelemek változásait végig vezesse a fejlesztőkörnyezetben lévő kódon. Például, ha egy típus adott műveletét átnevezzük, akkor a műveletre való összes hivatkozásban is megváltoztatja a korábbi nevet. Hasonlóan, ha egy osztályból kiemelünk egy interfészt, az interfész szerepelni fog az osztály megvalósított interfészeinek listáján.

Az IDE automatikusan felhívja a figyelmet azokra a lehetőségekre, amelyek refaktorálással segíthetnek a kód szerkezetét átalakítani, minőségét javítani. Például a 7-2 ábra azt mutatja be, hogy a Title és Publisher tulajdonságokat a C# ún. automatikus tulajdonság szintakszisával egyszerűbben is leírhatjuk.



7-2 ábra: Automatikus refaktorálási javaslatok

A szöveg melletti kis izzólámpa ikonra kattintva az eszköz végre is hajtja az adott refaktorálást (7-3 ábra)



7-3 ábra: A refaktorálás eredménye

A Visual Studio 2015 a .NET Compiler Platform segítségével a C# és Visual Basic projektekhez lehetővé teszi, hogy a fejlesztők saját analízis és refaktoring eszközöket írjanak (http://roslyn.codeplex.com/documentation).

A kód tesztfedettsége

Az alkalmazás elemeinek tesztelésére a csapat két különböző szemléletű automatikus tesztet is használ. Az ún. „fekete doboz tesztelés" (*black-box testing*) során csak a komponensek külső, kívülről elérhető felületeinek tesztelésére koncentrálnak. Az ún. „fehér dobozos tesztelés" (*white-box testing*) során pedig a belső működést a kód felépítése alapján tesztelik.

Józsefék tisztában vannak azzal, hogy sok esetben nincs arra lehetőségük, hogy minden lehetséges paraméterkombinációra felkészítsék a teszteseteket, egyszerűen azok nagy száma miatt.

Hogyan tudják mégis megmondani azt, hogy egy adott munkadarabhoz a hozzá tartozó tesztesetek is elkészültek? Ezt valójában soha nem lehet pontosan megmondani — hiszen nehéz definiálni, mit is értünk egyáltalán az összes teszteseten! Ugyanakkor léteznek hasznos technikák annak meghatározására, hogy vajon elegendő, a legtöbb fontos dologra kiterjedő tesztkészletünk van-e. Ezek közül a leggyakrabban használt — és az ellenőrző lista alkalmazás kis csapata is ezt használja — a tesztfedettség mérése.

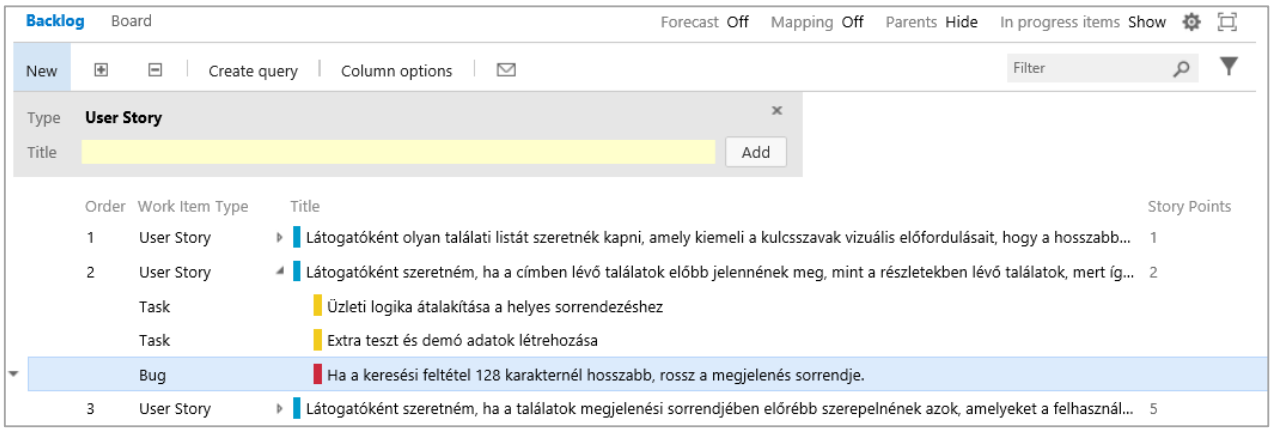
A tesztelés során a tesztfedettséget (*code coverage*) mérőszámot használhatjuk annak meghatározására, hogy vajon megfelelő mennyiségű tesztesetet hoztunk-e létre egy adott szolgáltatáselem ellenőrzéséhez. Ez a mérőszám azt mutatja meg, hogy a tesztek lefutása során a szolgáltatáshoz tartozó kód mekkora része (hány százaléka) került végrehajtásra. Ha ez az érték 100%, az azt jelenti, hogy egyetlen olyan kódsora sem volt a szolgáltatásnak, amely kimaradt volna a tesztelésből, tehát a csapatnak megfelelő magabiztosságot nyújthat ez a szám, mert valószínűleg alaposan végezte el az adott szolgáltatás tesztelését. Ugyanakkor, ha ez az érték – mondjuk – 30%, akkor ez bizony arra utal, hogy az adott szolgáltatás tesztelése távol jár az alapostól.

A tesztfedettség ugyan objektív szám, amely szoros korrelációban áll a teszteltség mértékével, azonban önmagában mégsem mond sokat a tesztelt rendszer minőségéről – és ezt nagyon fontos szem előtt tartani. Az, hogy egy adott szolgáltatás tesztelésének fedettsége 100%, még nem jelenti azt, hogy az jó minőségű, vagy akár azt, hogy a megfelelően alapos tesztek készültek el hozzá.

Regressziós tesztelés

Az alapos tesztelés ellenére előfordulhatnak olyan hibák, amelyekre a csapat nem készült fel. Ezeket lehet, hogy valamelyik felhasználó veszi észre, de az is lehet, hogy valamelyik csapattag. Ha a hiba az adott fejlesztési ciklusban lévő sztorira vonatkozik, akkor a sprint backlogba, az adott sztorihoz veszi fel a csapat, ha egy már használatban lévő — valamelyik előző sprintben leszállított — sztorihoz, akkor pedig a product backlogba kerül.

A 7-4 ábrán Tibor egy László által felfedezett hibát vesz fel a backlogba, amely arról szól, hogy .ha 128 karakternél hosszabb keresőszöveget ír be valaki, akkor nem a címben lévő találatok jelennek meg a találati lista elején.



7-4- ábra: Egy bug felvétele a backlogba

Bár kicsi a valószínűsége, hogy a felhasználók ezzel a hibával szembe találkoznak, Tibor azt beteszi a következő sprint backlogjába.

A következő sprint indulása után Józsefhez kerül a hiba javítása, aki először egy regressziós tesztesetet készít. Ez reprodukálja a hibát, és így a hozzátartozó teszt elbukik. József a teszt segítségével gyorsan megtalálja a hibát az egyik adatbázis-lekérdezésben, amelyik 128 karakterre vágja le a keresés kulcsát. A hiba javítása után a regressziós teszteset már helyesen lefut.

Egyéb tesztelési lehetőségek és minőségbiztosítási eszközök

A csapat az alapvető funkcionális tesztelés mellett természetesen még további tesztelést is végez. Az alkalmazáshoz készítenek egy gyors ellenőrző listát, amely segítségével manuális módon ellenőrizhetik, hogy annak felhasználói felülete megfelel-e az alapvető dizájn és ergonómiai elvárásoknak. Ennek az ellenőrző listának egy része a „kész" definíciójába is beletartozik, így a legfontosabb manuális ellenőrzéseket a csapat már a sztorik készre jelentése során elvégzi.

A kódminőség ellenőrzéséhez a csapatnak lehetősége van kódmetrikákat használni, illetve statikus kódanalízist végeznie.

8. Folyamatos termékleszállítás

Anita, József és László már saját korábbi tapasztalatából tudja, milyen kínos, amikor a rendszer használata során azt kell mondania: „de hát az én gépemen ez jól működik”! Hiába minden tesztelés a fejlesztői környezetben, ha az alkalmazás a demó és egyéb környezetekben valamiért nem fut! Az ellenőrző lista alkalmazást készítő csapat jól tudja, hogy nemcsak a teszteket célszerű automatizálnia, hanem azokat az eljárásokat is, amelyek a fejlesztőkörnyezetben tesztelt és működő kódokat megfelelő módon telepíti és konfigurálja az éles környezetekben is.

A folyamatos leszállítás olyan szoftverfejlesztési gyakorlat, amely a folyamat lépéseit igyekszik automatizálni azért, hogy ezzel javítsa a minőséget és a hatékonyságot. Több olyan alapvető technikából áll össze, mint például a korábban már bemutatott automatikus tesztelés, a folyamatos integráció (*continuous integration*) és a folyamatos terméktelepítés (*continuous deployment*). Ennek az eszköztárnak a használata gyors megoldásokat biztosíthat az újabb termékváltozatok ellenőrzött kibocsátására éppen úgy, mint a hibák felismerésére, javítására és a javítócsomagok telepítésére.

Folyamatos integráció

A folyamatos integráció fogalma már régóta ismert az agilis fejlesztés világában. Először az extreme programming (XP) vezette be, de azóta széles körben elterjedt, sőt még a vízesés alapú projekteken is gyakran használják ezt a technikát. Ennek az a lényege, hogy a szoftvertermék fejlesztésével foglalkozó csapat olyan forráskód követést használ, ahol a projekt fejlesztői által napi szinten előállított kódmódosításokat folyamatosan visszavezetik egy megosztott főágra. A kódbázis integrált kezelése mellett természetesen arról is gondoskodik a projekt, hogy a főágon lévő szoftver egyúttal a minőségi elvárásoknak megfeleljen.

A folyamatos integráció legfontosabb célja az agilitás alapgondolatának támogatása, ami szerint az előrehaladás elsődleges mércéje a működő szoftver. Legfontosabb elvei az alábbiak:

Használj forráskód követést!

Automatizáld a kódépítést!

Tedd a kódépítést önellenőrzővé!

Minden kódvisszaírás után végezd el újra a kódépítést!

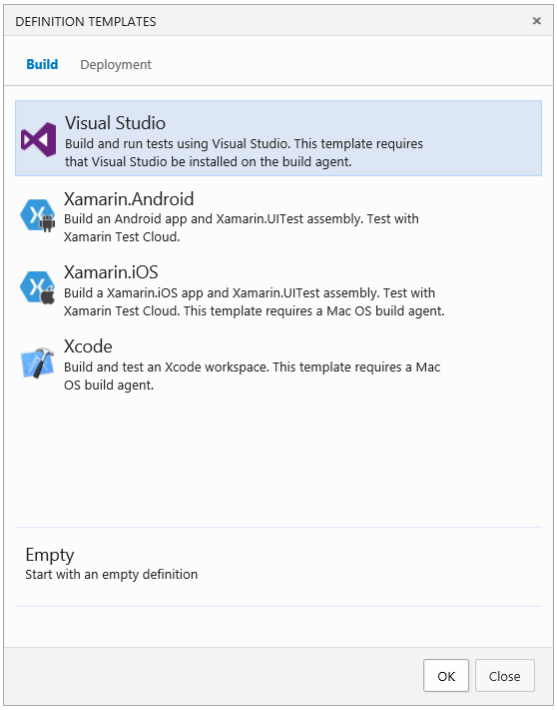
Gondoskodj arról, hogy gyors legyen a kódépítés!

Mindenki láthassa a legutóbbi kódépítés eredményét!

Kódépítés

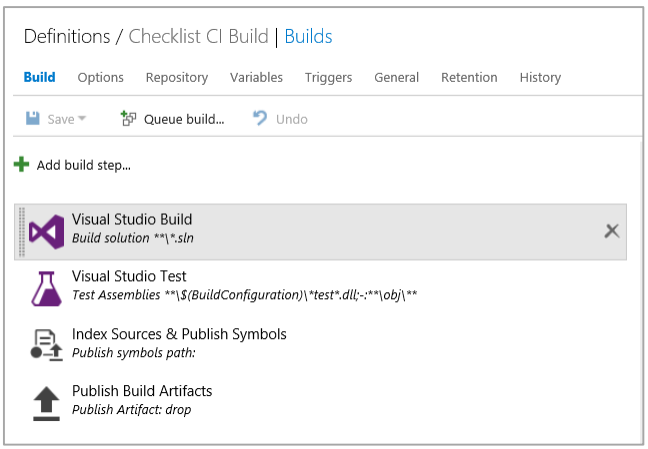
A csapat Anitára bízta a forráskód automatikus építésének elkészítését. Anita több kódépítő eljárást is elkészített. Ezek közül a legfontosabb a folyamatos integrációt megvalósító változat, amely gyorsan lefut. Anita egy olyan változatot is elkészített, amely rendszeresen (4 óránként) lefut, és ellenőrzi, hogy a kód fordítása után az nem csak a fejlesztői, hanem a demó környezetben is jól m űködik.

A kódépítést biztosító eljárások során a portálon felkínált sablonokból (8-1 ábra) indult ki, azok közül értelemszerűen a Visual Studio sablonját használta.



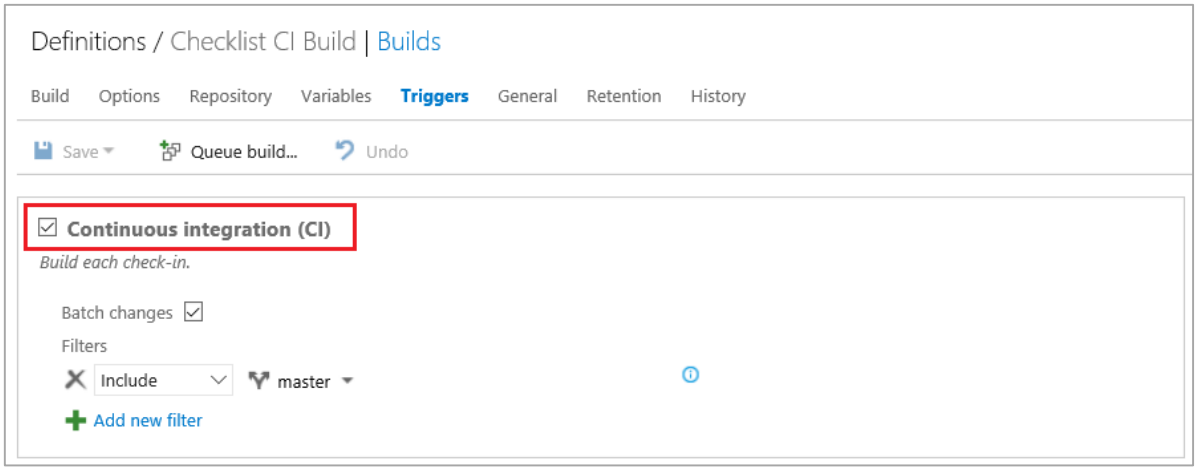
8-1 ábra: Kódépítés sablon választása

Anita a sablonban lévő kódépítési lépéseket az alkalmazásra szabta (8-2 ábra), így azok a kód fordítása mellett lefuttatják a teszteket, elkészítik a nyomkövetéshez használt szimbólumokat, és elérhetővé teszik a termék lefordított változatának elérését.



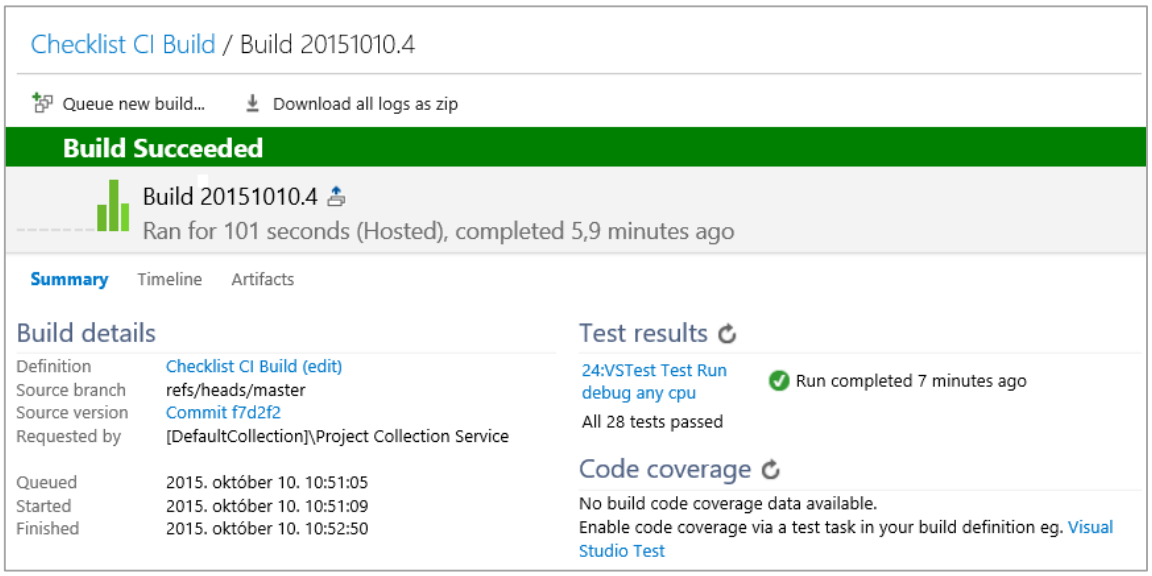
8-2 ábra: A kódépítés lépései

A folyamatos kódépítéshez szükséges opciót a Triggers fülön állította be, amint azt a 8-3 ábra mutatja. Ettől a pillanattól kezdve minden egyes kódvisszaírás során — amikor a kód a verziótárba kerül —, elindul a kódépítés folyamata és annak részeként az automatikus tesztek lefutása.



8-3: A folyamatos integráció bekapcsolása

A portálon a fejlesztőcsapat tagjai megnézhetik a kódépítés eredményét is (8-4 ábra).



8-4 ábra: A kódépítés eredményének összefoglalása

Folyamatos terméktelepítés

A folyamatos integráció gondoskodik arról, hogy észleljük azokat a kódolási hibákat, amelyek a stabil minőségű termékhez kapcsolódó elvárásokat megtörik, például funkcionális vagy minőségi hibákat vezetnek be, esetleg akár a kód sikeres lefordulását is megakadályozzák. Ugyanakkor a legtöbb fejlesztőcsapat számára ezen kívül még a termék éles környezetbe való telepítése is komoly kihívást jelent.

A legtöbb nehézséget az okozza, hogy a termékképességek folyamatos változtatása többfajta problémát is előidézhet, és ezek kezelése nem mindig egyszerű — és nem is minden esetben technológiai jellegű — feladat.

Ezekkel a nehézségekkel az ellenőrző lista alkalmazást fejlesztő csapat is tisztában van. Ők az alábbiakban látják a legfontosabb problémákat, amelyek a termékképességek változásából eredhetnek:

A felhasználói felület megváltozik — ez megzavarhatja a felhasználókat.

A felhasználói felület megváltozik — ez megzavarhatja a tesztelőket.

A korábban kiadott termék szolgáltatáshomlokzatai megváltoznak.

A termék mögött lévő adatbázis, illetve adatbázis-szerkezet megváltozik.

A termék konfigurációjá, paraméterei változnak egy új változat kibocsátása során.

Több termékváltozatot magában foglaló „ugrást” kell az éles környezetben végrehajtani.

Bár az alkalmazás jelenlegi fejlesztési ciklusaiban még a csapatnak nem kell ezekkel a problémákkal számolnia, mert egyelőre nem ad ki az ellenőrző lista alkalmazásból publikusan elérhető változatokat, azért készülnek arra, hogy előbb vagy utóbb foglalkozniuk kell ezekkel a kérdésekkel. József, Anita és László Tiborral együtt az alábbi teendőket határozták meg a folyamatos termékleszállítás megfelelő kezelésére:

Kódépítés és folyamatos integráció biztosítása. A folyamatosan változó kódból a csapatnak bármikor elő kell tudni állítani az ellenőrzött, telepíthető termékváltozatot.

Manuális tesztelés. Az automatikusan nehezen vagy csak nagyon költségesen tesztelhető funkciók ellenőrzéséhez (felhasználói felület, *end-to-end* tesztelés, esetleg teljesítménytesztelés) a csapat kézi tesztelést fog bevezetni.

Kibocsátás és konfiguráció kezelése. Amikor már többfajta mobil eszközön is használni fogják az alkalmazást, a csapat követni fogja, hogy az adott eszközökre milyen konfigurációban telepítették ki az alkalmazás egy adott változatát.

9. A termék működésének követése

A termék kezelése során a csapat nem elégedhet meg azzal, hogy egyszerűen csak egy jó minőségű alkalmazást bocsát ki. Folyamatosan figyelnie kell a felhasználók visszajelzéseire. Ennek egy jó módja, hogy az alkalmazáshoz tartozó értékelő megjegyzéseket elolvassák, esetleg időnként egyszerű kérdőívekkel felmérik azt, hogy a felhasználók milyen új funkcióknak örülnének. Hasonlóan jól használhatók az esetleges hibabaejelentések, támogatási kérések.

Ezek a módszerek kétségtelenül hasznosak, de nem tudnak mindenben választ adni a fejlesztőcsapat esetleges kérdéseire. Mely funkcióikat használják a leggyakrabban? Melyik funkció az, amelyet a felhasználók leggyakrabban félbehagynak? Az alkalmazás mely részéről érkeznek olyan hibajelzések, amelyek a felhasználó (esetleg a rendszer) hibájára utalnak?

Még ha ezeket a kérdéseket meg is tudják válaszolni az alkalmazás felhasználói, egy kérdőíves kutatás akkor sem lehet igazán jó ezeknek az információknak a begyűjtésére. Nem tudunk minden felhasználót elérni, illetve a kérdőívek kitöltésével és kiértékelésével hosszú idő telhet el.

Használati analitika

Az azonnali visszajelzések egyik legjobb módja az, ha az alkalmazást készítjük fel arra, hogy a használat módjáról adatot gyűjtsön. Ennek legegyszerűbb módja a rendszerek szerver oldalán lévő műveletek naplózása, amelyből kiderülhet, hogy melyek a gyakrabban és ritkábban használt funkciók, hol következnek be általában felhasználói vagy rendszerhibák.

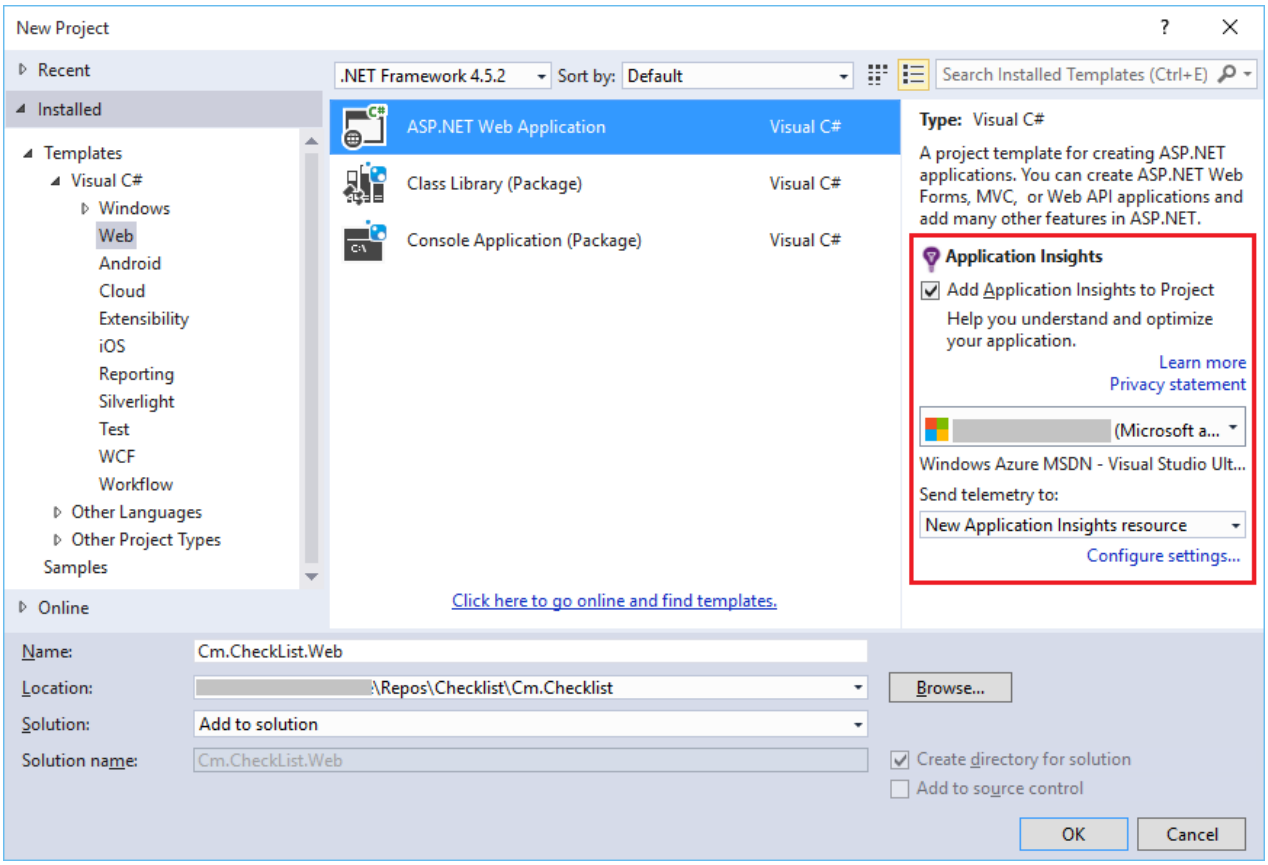
Annak okát, hogy az alkalmazás néhány olyan funkcióját, amelytől sokkal intenzívebb használatot vártunk, mégsem veszik birtokba a felhasználók, a szerveroldali naplózás nem segít kideríteni. Egy jó alkalmazásban a begyűjtött információ segítségével követhetjük a felhasználók útját, ahogyan azok a rendszer felületén a funkciók között vándorolnak. Összegyűjthetjük, hogyan haladnak végig képernyőkön, hol, mennyi időt töltenek, mikor szakítanak meg egy elkezdett folyamatot még annak befejezése előtt.

Egy ilyen naplózott folyamat (*user journey*) segítségével elemezhetjük a felhasználók viselkedését, és megkereshetjük azokat a megoldásokat, amelyekkel ráirányíthatjuk a figyelmüket a fontosnak ítélt funkciókra, segítünk nekik gyorsabban rátalálni azokra. Egy elemzés rámutathat azokra a termékképességekre is, amelyekre nincs valós igény — ezeket akár el is távolíthatjuk a termékből.

Application Insights használata

Az alkalmazások monitorozásának egyik legegyszerűbb eszköze a Microsoft Azure szolgáltatásainak egyike, az Application Insights, amely lehetővé teszi a használat nyomon követését, a hibák észlelését és a szoftverleállások diagnosztizálását.

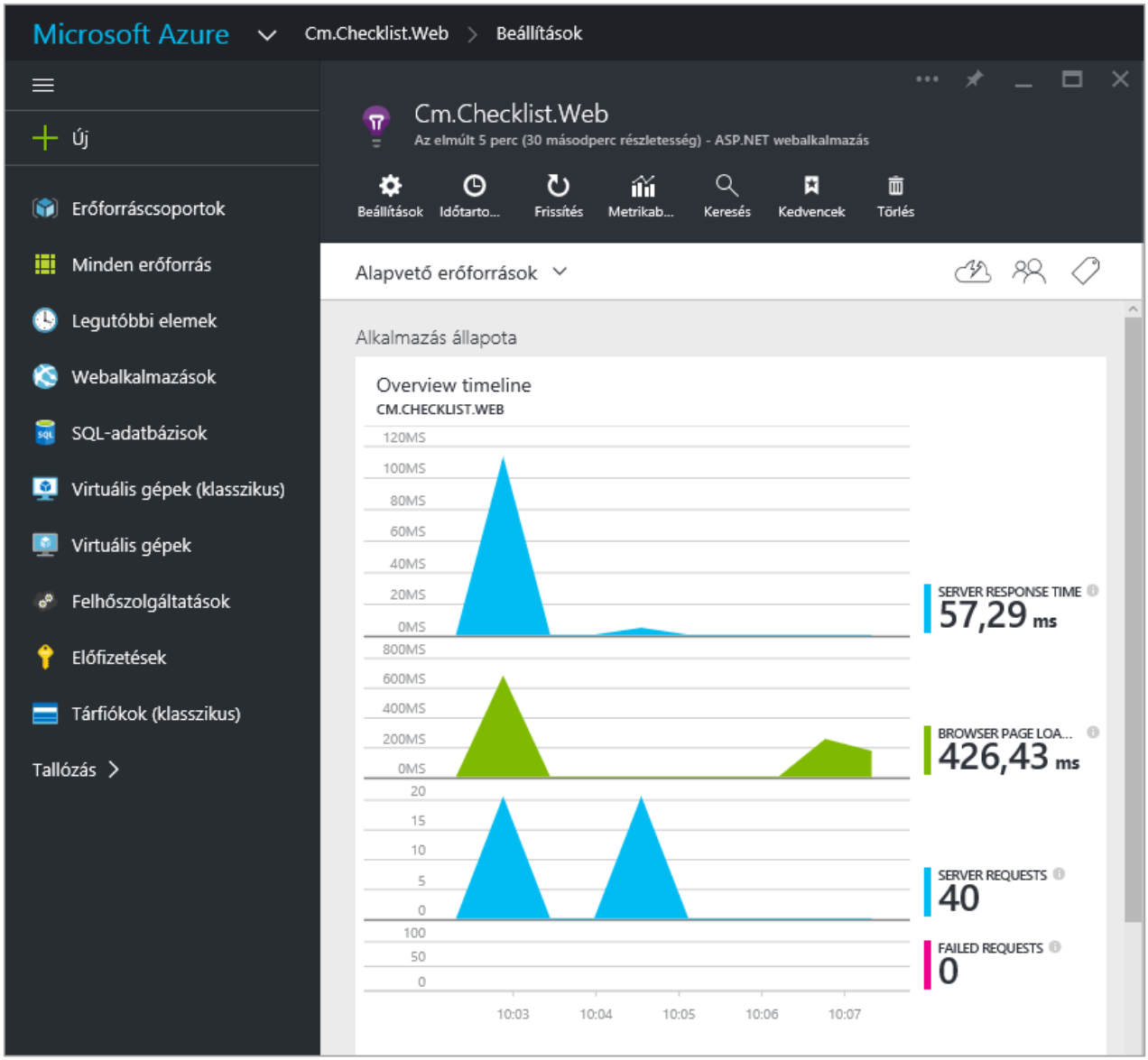
Tibor fontos feladatnak tartja a felhasználási információk elemzését, ezért a csapattal együtt a következő sprintre definiál egy deszkamodellt az Application Insight megismerésére. József a feladathoz egy olyan webes projektet hoz létre, amely integrálja a diagnosztikát (9-1 ábra)



9-1 ábra: Új webalkalmazás létrehozása Application Insigts integrációval

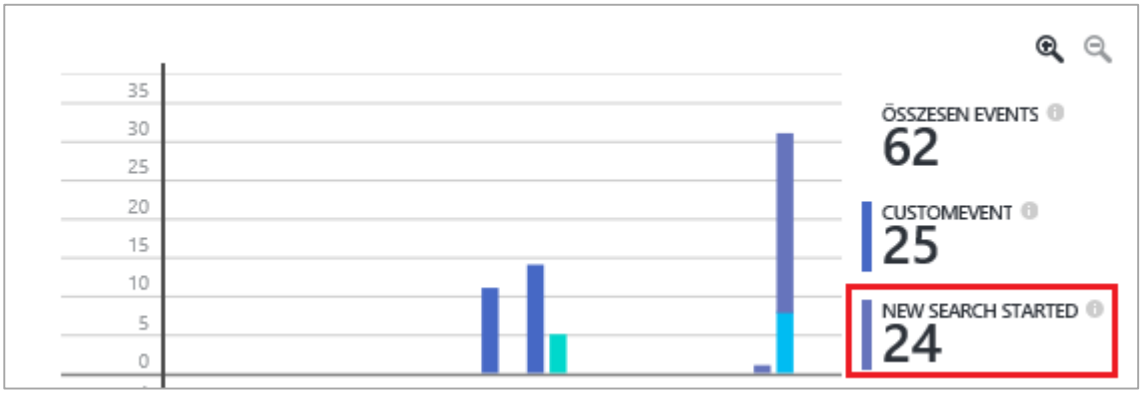
A projekt létrehozása után a diagnosztika azonnal működőképes. József néhány alkalommal elindítja az alkalmazást és navigál annak lapjai között. Ennek az egyszerű szimulációnak az eredményeképpen azonnal megtekintheti az Application Insight által összegyűjtött információt (9-2 ábra).

9. A termék működésének követése



9-2 ábra: Néhány alapvető diagnosztikai információ

Józsefnek nem kell megelégednie a rendszer által automatikusan gyűjtött diagnosztikai adatokkal. Próbaképpen beépíti az alkalmazás kliens oldalába annak naplózását, hogy a felhasználó egy új keresést indít. Ehhez egyetlen JavaScript utasításssorra van szüksége, és az adatokat azonnal leolvashatja az alkalmazáshoz tartozó AppInsight portálon (9-3 ábra).



9-3 ábra: Alkalmazás-specifikus események

Visszacsatolások kezelése

Az információk összegyűjtése és elemzése során a csapat kialakíthatja azt a képet, hogy milyen termékképességeket kell megváltoztatnia, illetve milyen új képességeket kell a termékhez adnia, hogy az jobban segítse őket — és felhasználóikat — a céljaik elérésében. Természetesen ennek kitalálása nem egyszerű feladat.

Ezeket a módosításokat és új termékképességeket a csapat a product backlogba írja új sztorik formájában. A termékgazda, Tibor feladata azok priorizálása és további egyeztetése a csapattal, hogy az értékes termékképességek minél hamarabb beépítésre kerüljenek.

Összegés

A gyors termékfejlesztés ma már csak valamelyik agilis szemléletet támogató módszer vagy keretrendszer segítségével lehetséges. Az agilis szemléletmód nem a programozói — kódolói — hatékonyság növelésével tesz termelékenyebbé egy fejlesztőcsapatot, hanem az alábbiak támogatásával:

Az értékes funkciók fejlesztésére fókuszál, azokat tekinti elsődleges prioritásnak.

Nagy csapatok, hosszú idő és tengernyi munka helyett kis csapatban, rövid ciklusidőben és apró munkadarabokban gondolkodik.

A termék megvalósítása során a csapat folyamatosan alacsony szinten tartja a műszaki adósságot, vagyis nemcsak az alkalmazás önmagában vett kiváló minőségére, hanem annak egyszerű karbantarthatóságára is törekszik.

A termék elkészült — akár apró — képességeiről is azonnali felhasználói visszacsatolást szeretne kapni, az észrevételekből leszűrt tapasztalatokat azonnal igyekszik megjeleníteni a termékben.

A Visual Studio Professional 2015 — önmaga is agilis szemléletmóddal készül — minden tevékenységben támogatja azokat a fejlesztőket és csapatokat, akik ezt a megközelítésmódot alkalmazzák, illetve alkalmazni tervezik. A fejlesztői gépre telepített IDE a Visual Studio Team Services[3] és a Microsoft Azure szolgáltatásaival együtt minden fontos termékképességet tartalmaz, amellyel a csapatok alkalmazásaikat termékként fejleszthetik.

[3] Korábbi nevén: Visual Studio Online

Visual Studio

A Visual Studio a Microsoft integrált fejlesztői környezete, amely több termékváltozatban is elérhető. A Visual Studio önmagában illetve MSDN előfizetéssel is megvásárolható.[4]

Mi az az MSDN?

Egy Visual Studiohoz kapcsolt előfizetés, amely az alábbi előnyöket biztosítja:

- Szoftverfrissítési Garancia – újabb verziók folyamatos elérhetősége a licencelési időszak alatt.
- Fejlesztés-tesztelés céljára további Microsoft szoftverek és havi Azure krediteket.
- O365 Developer előfizetés.
- A fejlesztő számára Team Foundation Server (TFS) + TFS Call licenceket.
- Technikai support incidens kezelésből 2 db.
- Windows és Windows Phone developer account.

Az MSDN-nel kapcsolatban a legfontosabb, hogy 1 MSDN előfizetést csak 1 emberhez lehet hozzátársítani.

Melyek az aktuális Visual Studio verziók?

Visual Studio Enterprise with MSDN
Visual Studio Professional with MSDN
MSDN Platforms
Visual Studio Test Professional with MSDN
Visual Studio Professional 2015

Az egyes verziókat részletesen összehasonlító oldalért látogasd meg a Visual Studio hivatalos oldalát.

Hogyan lehet megvásárolni?

Keresd a helyi viszonteladódat, vagy látogasd meg a Visual Studio hivatalos oldalát.

Ha csak ki akarod próbálni,

akkor vedd igénybe a díjmentes Visual Studio Dev Essentials program szolgáltatásait az alábbi oldalon: http://my.visualstudio.com

Ha kérdésed van, vagy segítségre lenne szükséged írj az alábbi email címre: devtools_hun@microsoft.com

[4] Ez az oldal a 2015.11.18-i állapotot írja le.