Budapesti Corvinus Egyetem

Gazdaságtudományi Kar

Számítástudományi Tanszék

**Forrás Hivatkozások számonkérésének automatizálása**

Készítette: Mátrai Tibor

Gazdaságinformatikus

2021

Burka Dávid

Tartalom

[1. Bevezetés 2](#_Toc69568754)

[2. Projekéletciklus modellek vizsgálata 3](#_Toc69568755)

[2.1 Vízesésmodell 3](#_Toc69568756)

[2.1.1 Követelmények 4](#_Toc69568757)

[2.1.2 Tervezés 4](#_Toc69568758)

[2.1.3 Implementáció/kódolás 5](#_Toc69568759)

[2.1.4 Tesztelés 5](#_Toc69568760)

[2.1.5 Támogatás és karbantartás 6](#_Toc69568761)

[2.1.6 A vízesés modell előnyei és hátrányai 6](#_Toc69568762)

[2.2 Agilis életciklus modellek 7](#_Toc69568763)

[2.2.1 Extreme programozás 7](#_Toc69568764)

[2.2.2 Scrum 9](#_Toc69568765)

[3. Irodalomjegyzék 11](#_Toc69568766)

# 1. Bevezetés

A szakdolgozatom témájának a Budapesti Corvinus Egyetem Gazdaságinformatikus hallgatóinak a szakdolgozatukhoz tartozó hivatkozási szabályok számonkérésének az automatizálását választottam. Az ötlet akkor született amikor először kellet ezt a számonkérést megírnom, és az akkori szeminárium vezetőm panaszkodott arra, hogy milyen hosszú folyamat az akkori számonkérési feladatok javítása. Abban az évben a zárthelyi dolgozat úgy épült fel, hogy a hallgatók kaptak egy másfél-két oldalas Word fájlt irodalom jegyzékkel együtt, amely tartalmazott 10 hibát a hivatkozásaiban. A hallgatók feladata ezen hibák megtalálása és javítása. A csalások megakadályozása érdekében az egy teremben írók között négy féle feladatsort osztottak ki.

A dolgozat célja meghatározni, hogy lehetséges és megéri-e ezt a számonkérést automatizálni valamilyen módon, és fejleszteni egy olyan applikációt, ami ezt lehetővé teszi, és ezzel időt spórolni a javításon, ami egyrészt pénzt spórol az egyetemnek és megkönnyíti a hallgató életét azzal, hogy előbb tudják, hogy hogy sikerült számukra a számonkérés. A feladat tehát egy olyan applikáció fejlesztése, amely megkönnyíti a feladatsorok összeállítását és valamilyen módon meggyorsítja a javítást is.

A dolgozat második részében elsőként megvizsgálom a különböző számonkérési típusokat, melyeknek milyen pedagógiai hatásai vannak, hogyan lehetne őket automatizálni, és hogyan felelnének meg ezen szabályok elsajátításának a számonkérésére. Ez utána az applikáció fejlesztésének megkezdése elött meg kell tervezni a projektnek az életciklusát. Tanulmányaim, munkám és ezen dolgozat megírása során sok életciklus modellt ismertem meg, és a projekt megkezdése elött elemeztem, hogy számomra melyek felelnének meg. Elsőként a vízesés modell történetét, működését, megítélését és problémáit fogom bemutatni. A második részben pedig az agilis életciklus modellekkel fogok foglalkozni, elsőként bemutatom az álltalános történetüket és kialakulásukat, majd részletesen kitérek két modellre az extrém programozási modellre és a Scrumra. Elemzem ezen modellek működését, alappilléreit, megítélésüket és végül azt, hogy milyen esetekben érdemes ezeket választani A második rész végén pedig bemutatom a fejlesztés során használt keretrendszereket.

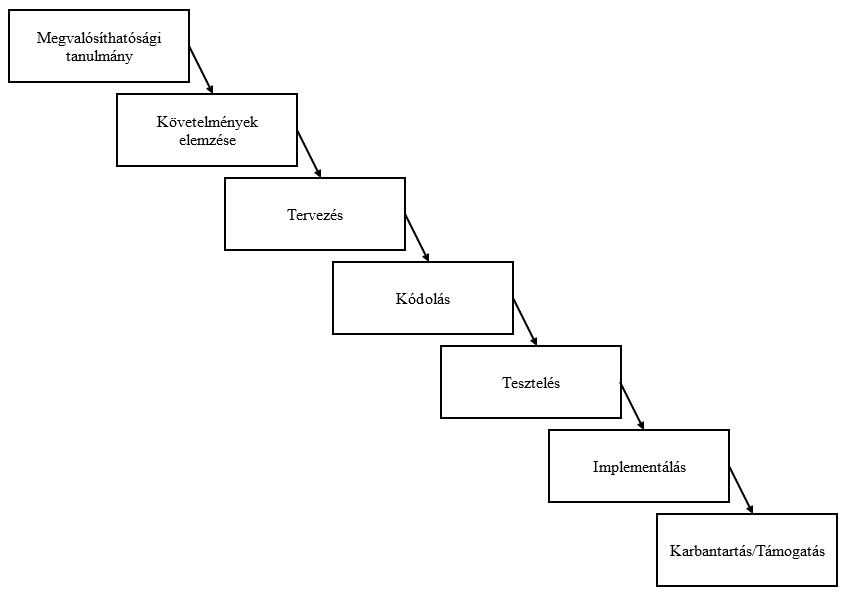
A dolgozat harmadik részében bemutatom a projektemet. Először leírom az oktatókkal közösen összeállított követelmény listát és megindokolom, miért ezt a számonkérési módszert választottam. Utána szó lesz a projekt közben alkalmazott életciklus modellről, a tervezés, a fejlesztés és a tesztelés lépéseiről. Majd az elkészült alkalmazás belső felépítéséről és a felhasználói felület kialakításáról. Később pedig foglalkozni fogok a kész alkalmazás használatának bemutatásával és a karbantartásának a feladatával és esetleges problémáival.

A szakdolgozatom utolsó részében összefoglalom a dolgozatban foglaltakat és megválaszolom a feltett kérdést, hogy érdemes-e ennek a feladatnak ilyen módú automatizálásával foglalkozni.

# 2. Projektéletciklus modellek vizsgálata

## 2.1 Vízesésmodell

A vízesésmodell volt a tradicionális módja a kis és nagy informatikai projekteknek a kilencvenes évektől egészen a kétezer-tízes évekig (Dima - Maassen, 2018), a 2018-ban készült felmérés szerint a megkérdezett szakértők 32% még mindig ezt használta (Dima - Maassen, 2018). Ez a modell alapvetően 5 elemből áll, de némely forrás 6-ot, illetve 7-et említ, ezeket a lépéseket a következőkben fogom részletezni. Ezek a lépések akkor követik egymást, ha az előző már teljesen befejeződött, amennyiben egy korábbi lépésen szeretnénk módosítani akkor azon lépés után következő összes lépést újra el kell végezni (Stober – Hansmann, 2010).

Az 1-es ábrán a Chandra (2015) megfogalmazott 7 lépés látható a két plusz elem Stober és Hansmann (2010) által megfogalmazotthoz képest a megvalósíthatósági tanulmány és a tesztelés utáni implementáció. A következőkben viszont Stober és Hansmann 2010-es könyvében meghatározott öt fázis (Követelmények, Tervezés, Implementáció/Kódolás, Tesztelés, Támogatás) szerint fogok haladni és közben kitérek arra, hogy ők ezeket a lépéseket mikor végzik el.

1. ábra vízesésmodell (forrás: Chandra, 2015)

### 2.1.1 Követelmények

Stober és Hansmann (2010) szerint a vízesésmodellel végrehajtott projektek első fázisának neve a követelmények. Ezt a fázist több részre bontanak, első lépésként azonosítják a projekt részvevőit és elemzik, hogy kinek mekkora a beleszólása a projektbe. Csak ezután kezdik el begyűjteni a résztvevők igényeit. Ez azért fontos, hogy pontosabb képet kapjanak a következő lépésen dolgozok hiszen, ha ebben a szakaszban rosszul fogalmazzuk meg a követelményeket az majd csak az utolsó lépéseknél fog kiderülni, és egy változtatást akkor bevezetni már nagyon költséges, erről majd a későbbiekben bővebben is lesz szó.

A résztvevők azonosítása után megkezdődik az igények felmérése, begyűjtése. Az igényekből egy elemzés során lesznek követelmények melyeket két fő csoportba sorolnak, a funkcionális és a nem funkcionális követelmények közé (Stober – Hansmann, 2010). Ezeket a követelményeket dokumentálják és adják tovább a következő fázisnak.

Chandra (2015) ezt a fajta követelmények fázist veszi ketté, első lépésként begyűjti a követelményeket, amely az 1. ábrán megvalósítgatási tanulmányként szerepel. Ebben a részben megnézi, hogy a begyűjtött követelmények közül melyik megvalósítható a rendelkezésre álló ezközökkel. Az 1. ábrán látható második lépésként pedig elemzi a követelmény listát és ez után pedig dokumentálja azokat.

### 2.1.2 Tervezés

A tervezés fázisban a tervező csapat egy részletes tervet készít az egész rendszerről és minden egyes komponensről. Ez olyan részletességgel történik, hogy ezeket a komponenseket a fejlesztők már rögtön kóddá tudják alakítani (Stober – Hansmann, 2010). Ennek a résznek az eredményét Chandra (2010) a kódolás alaprajzának nevezi. Több módszer is van ennek az „alaprajznak” az elkészítésére, Stober és Hansmann 2010-ben hármat említett.

Elsőként a használati eset-modellt (Use Case-model), amely az előző fázisban dokumentált követelményeket bontja fel és részletezi őket olyan mélységig, hogy a rendszer összes interakciójára szerepeljen egy külön használati eset. De lehetnek magasabb szintű esetek, amelyeket később bontanak majd csak le több külön esetre. A használati esetek a tervezés folyamat végén egy táblázatba kerülnek, ahol álltalában egy eset egy sor, de mint az előbb említettem lehet, hogy egy eset későbbi felbontás után több sort is elfoglal. A táblázat oszlopai tartalmazhatják az eset nevét azonosító számát, verzióját (a lehetséges módosítások követése miatt), várt eredményét, íróját és még sok más adatot, ez attól függ, hogy milyen részletes a használati eset-modell írója (Stober – Hansmann, 2010).

A második módszer az úgynevezett Unified Modeling Language (UML), ami egy szabványos, álltalános célú modellező nyelv az objektumokat tudják jól ábrázolni az objektum orientált programozáshoz. A harmadik mód pedig folyamat ábra segítségével történik, ahol a követelmények alapján készül a folyamat ábra, amely tartalmazza a program által összes bejárható lépéseket (Stober – Hansmann, 2010).

Chandra (2015) is ugyanezeket a modelleket sorolta fel.

### 2.1.3 Implementáció/kódolás

Az implementáció fázisban készül el a fejlesztők által a program kód (Chandra, 2015). Az előző fázisból megkapott modell segítségével a programozók megírják a tényleges szoftvert. Ebben a szakaszban derülnek ki a problémák, ha nem volt megfelelően elvégezve a tervezési szakasz. Amennyiben itt hibát találnak akkor vissza kell menni az előző szakaszba és újra kell tervezni a folyamatokat vagy használati eseteket. Ebben a szakaszban is zajlik már tesztelés, ezt fejlesztői vagy integrációs tesztnek nevezzük. Itt csak azt akarjuk biztosítani, hogy a tesztelő csapatnak nem lesznek alap problémái, és ők fókuszálhatnak az összetettebb tesztesetekre (Stober – Hansmann, 2010). Ha egy hibát csak a tesztelők találnak meg azt 10-15-ször annyiba kerül majd kijavítani, mint ha már a fejlesztő megtalálná (McConnel, 2004).

### 2.1.4 Tesztelés

Ez előbb említett integrációs teszt sikeres lefutása után kezdődik meg a teszt csapat általi tesztelés. Ennek a célja megtalálni az összes hibát az adott rendszerben a kiadás vagy átadás elött (Stober – Hansmann, 2010). Ez egy lehetetlen feladat ezért a tesztelők álltalában addig tesztelnek amig elérnek egy becsült százalékot a tesztesetek között vagy addig amíg a talált hibák az eltelt idő alatt egy ellaposodó tendenciát mutatnak (Stober – Hansmann, 2010).

A tesztelés szakaszának tartalmazni kell különböző teszt típusokat. Ezek a szoftver különböző elemeire koncentrálnak. Ezek a tesztek lehetnek: integrációs és funkcionális tesztek, globalizáció ellenőrző teszt, fordítás ellenőrző teszt, rendszer ellenőrző teszt, teljesítmény teszt, elfogadási teszt (Stober – Hansmann, 2010).

A tesztelés sikeres elvégzése után átadják a kész szoftvert a megrendelőnek vagy piacra bocsájtják. Ezt a műveletet Chandra (2015) külön szakaszba sorolja implementáció néven mivel ekkor implementáljuk a szoftvert a megrendelő rendszerébe.

### 2.1.5 Támogatás és karbantartás

A támogatás fázis egyből elkezdődik amint átadják a szoftver a megrendelőnek. A támogatás több szintű lehet, vannak esetek amikor csak hívás központon keresztül segítenek a felhasználónak. Komolyabb problémáknál egy specialista segíthet a probléma megoldásában. A legkomolyabb esetben pedig a fejlesztők javítják ki a talált hibát a rendszerben (Stober – Hansmann, 2010). Az innen érkező hibák javítása akár 10-100-szorosa is lehet annak mintha már az implementációs fázisban megtalálták volna (McConnel, 2004).

### 2.1.6 A vízesés modell előnyei és hátrányai

A vízesésmodell legnagyobb előnye, hogy akkor a leghatékonyabb a projekt végre hajtása, ha az elején mindent megtervezünk és egy komplett befejezett követelmény listával dolgozhatunk (Stober – Hansmann, 2010). A vízesésmodellnek pedig ez a lényege, hogy addig nem állunk hozzá a tervezéshez ameddig a követelmények fázis nem teljes. A modell következő nagy előnye, hogy jól definiáltak a fázisok (Chandra, 2015), és ennek köszönhetően könnyen megmondható, hogy hol tart a projekt, és könnyen becsülhető a hátra lévő idő is feltéve, hogy nem lépnek fel hibák. Előnye továbbá, hogy a kódolási és implementációs fázis egyszerűen megvalósítható (Chandra, 2015), köszönhetően a korai részletes tervezésnek és annak, hogy a folyamat végrehajtása közben nem érkeznek új követelmények. A Stober és Hansmann (2010) féle követelmény gyűjtés során végzett résztvevő azonosításnak köszönhetően, Chandra (2015) felsorolja előnyként azt, hogy a projekt során jól ismerjük a vég felhasználókat és ez által személyre szabottabb végterméket adhatunk át.

A vízesésmodell fő hátránya, amit Chandra (2015), Dima és Maassen (2018), Stober és Hansmann (2010) is említ az a rugalmasság hiánya. Ez Stober és Hansmann (2010) által megfogalmazott szabályból is következik, mi szerint addig nem léphetünk a következő fázisba ameddig az előzőt teljesen el nem végeztük. Az ezzel a modellel végre hajtott projektekben nem lehet vagy nagyon költséges közben követelményt változtatni hiszen ebben az esetben elölről kell kezdeni a tervezési fázist. Nem tud visszajelzés érkezni a megrendelő felől (Chandra, 2015), hiszen neki csak kétszer van betekintése a projektbe, a követelmény és az átadási fázisokban (Stober – Hansmann, 2010).

A vízesésmodellt mára már primitívnek nevezik, de rövid egyszerű projektek elvégzésére még mindig a leghatékonyabb eszköz (Chandra, 2015).

## 2.2 Agilis életciklus modellek

A vízesés modell rugalmatlansága miatt megjelent az igény új fajta projekt életciklus modellekre. 2001-ben kezdődött meg az agilis alap modellek kialakulása, az agilis manifesztum megjelenésével (Abrahamsson - Salo - Warsta, 2002). Ez a kiállítmány négy fő kijelentésből állt:

1. „Az egyének és a személyes kommunikáció a módszertanokkal és az eszközökkel szemben”
2. „A működő szoftver az átfogó dokumentációval szemben”
3. „A rendelővel történő együttműködés a szerződéses egyezéssel szemben”
4. „A változtatás iránti készséget a tervek szolgai követésével szemben”

A felsorolt négy pontban mindig az első kijelentést tartják fontosabbat a második helyett (Abrahamsson et. al. 2002, 11 old.). A második kijelentések az előbb bemutatott vízesésmodellre utalnak.

Jelenleg többféle módszertan van agilis fejlesztésre, mindnek meg van az előnye és a hátránya. a 2018-ban Dina és Maassen által készült kutatás során a válaszolók 68 százaléka használt valamilyen agilis módszertant közülük 77 százalék használt Scrumot, 15 százalék tesztvezérelt fejlesztést és a maradék 8 százalék pedig nem említette, hogy milyen módszertant használ. Míg a Szabó Bálint és Ribényi Máté által 2018-ban készült kutatása alapján az ott megkérdezettek 94 százaléka dolgozott már olyan projekten, ami Scrum módszertannal vezettek végig, 65 százalékuk kanbant és 40 százalékuk pedig eXtreme programozást használt a munkája során. Ezekből az adatokból kiindulva és történelmi jelentősége miatt, a következőkben az eXtreme programozást és a Scrumot fogom bemutatni.

### 2.2.1 Extreme programozás

Az eXtreme programozást először 1999-ben említette meg Kent Beck az egyik könyvében (Stober – Hansmann, 2010), tehát már az agilis manifesztum kiadása elött. Ennek a modellnek tizenkét alap elgondolás az alapja, amelyeknek megnevezése más lehet a forrásoktól függően, de lényegben ugyan azok. Ezek több forrás által is említettek a Páros programozás, az egyszerű kódra törekvés, a kiadások során törekvés a kis mennyiségű újdonságra, a folyamatos tesztelés, folyamatos kapcsolat a megrendelővel és a story cardok használata (Abrahamsson et. al. 2002, Stober – Hansmann, 2010, Szabó - Ribényi, 2018). A story cardok hasonlóak az előző részben említett use case-ekhez, azonban ezek teljes funkciókat tartalmaznak.

Az eXtreme programozás folyamata Abrahamsson et. al. (2002) szerint öt lépésből áll. Ezek rendre: feltérképezés, tervezés, iteráció és kiadás, termékizálás, karbantartás és a termék halála. Feltérképezés fázisban a megrendelő összeírja story cardokra az általuk megvalósítani kívánt funkciókat és követelményeket, és közben a fejlesztő csapat ismerkedik a keretrendszerekkel és a technológiákkal.

A tervezés fázisában prioritás szerint sorba rendezik a story cardokat, ezen a sorrenden a megrendelő ezután változtathat. Az első tervezés feladata kiválasztani azokat a követelményeket a story cardokról amelyek nélkül nem lehet működő kiadást készíteni (Abrahamsson et. al. 2002).

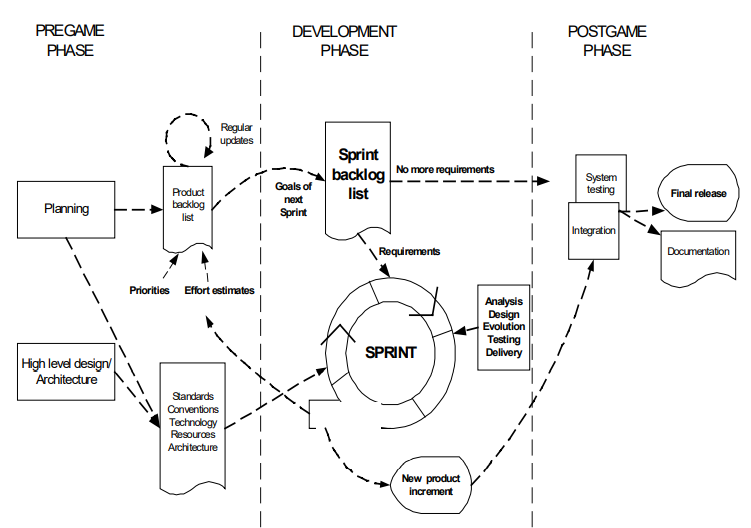
A tervezés után kezdődnek az iterációk és kiadások, az eXtreme programozás során az első kiadás előttiek kivételével, minden iteráció egy story card tartalmával foglalkozik, azaz minden iteráció egy követelménnyel vagy funkcióval foglalkozik. Az első kiadás elött azért nem mert az itt keletkezett program sorok magukban még nem képeznek működő alkalmazást. Ezt követően viszont minden iteráció után egy működő programnak kell elkészülnie, amelynek készen kell lennie éles használatra. Ennek a fázisnak része az adott iterációban újonnan hozzáadott funkció tesztelése is, a tesztesetek megírása a megrendelő feladata (Abrahamsson et. al. 2002).

A következő fázis feladata az elkészült kiadás további tesztelése, hogy az tényleg átadhassuk a felhasználóknak. Itt is adhat hozzá még a megrendelő új követelmény ezeket már gyorsabban kell végre hajtani, mint az előzőket, ezek az igények a tervezés folyamata során készülnek bele a prioritás listába és onnan fejlesztés alá (Abrahamsson et. al. 2002).

A karbantartás fázis hasonló, mint a vízesésmodellnél azzal a különbséggel, hogy itt még adhatnak hozzá új funkciót. Az eXtreme projekteknek a vége a halál fázis. Ez akkor történik amikor a megrendelő már nem akar új funkciót hozzáadni, ekkor készítik el a teljes projekt részletes dokumentációját és teljesen átadják a megrendelő cégnek (Abrahamsson et. al. 2002).

Az eXtreme programozás kis és közepes fejlesztő csoportoknak alkalmas, és csak olyan megrendelők esetében, akik képesek rendszeres visszajelzésre (Abrahamsson et. al. 2002). Nem ajánlott nagy projektekhez, ebben az esetben nem alkalmazható az alap elgondolások közül több sem, mint például a kis kiadások, az egyszerűség, és a tervezés egyszerűsítése (Stober – Hansmann, 2010).

### 2.2.2 Scrum

A Scrum keretrendszert először 1986-ban került dokumentálásra Takeuchi és Nonaka cikkében, ahol egy adaptív, rendszerező, és gyors termék fejlesztési eljárást írnak le. Ezt az eljárást 2002-ben írta át Schwaber és Beedle informatikai fejlesztési projektekre. A Scrum modell arra koncentrál, hogy a fejlesztő csapatnak hogyan kell együtt dolgoznia úgy, hogy képesek legyenek megbirkózni a folyamatosan változó környezeti tényezőkkel (követelmények, idő keret, erőforrások, technológia) (Abrahamsson et. al. 2002). 

2. ábra Scrum folyamati ábra forrás: Abrahamsson et. al. (2002)

Abrahamsson et. al. (2002), ahogy a 2. ábrán látható, három részre bontja a scrum folyamatát, a pregame-re, a fejlesztésre, és a postgame-re. Az első részt tovább bontják két al-fázisra, melyek a tervezés és az architektúra. A tervezés során elkészül a Product backlog listát, amely tartalmazza az összes ekkor ismert követelményt. Ez a lista folyamatosan bővül a projekt során a hozzá adott követelményekkel. Ebben a szakaszban határozzák meg a projekt során használandó szerepeket is. Ez után következik az architektúra szakasz, amely során készül egy magas szintű terv a programról a rendelkezésre álló követelmények alapján.

A pregame fázis után kezdődik a fejlesztés. Ebben a szakaszban úgy nevezett sprinteket végeznek (ahogy a 2. ábrán látható). Minden sprint tartalmazza a vízesésmodellben már leírt szakaszokat (Követelmények, Tervezés, Implementáció/Kódolás, Tesztelés, Átadás), egy sprint egy hét és egy hónap közötti hosszúságú lehet. A sprintek során minden napelején van egy tíz-tizenöt perces stand-up meeting, amely során mindenki elmondja mit csinált előző nap és mit tervez ma, ezzel elősegítve a csapaton belüli kommunikációt. A spint része továbbá a sprint elején elvégzett úgy nevezett sprint tervezés (amely során kiválogatják, hogy mit is fognak csinálni ebben a sprintben), a végén pedig egy review található (ahol megbeszélik, hogy mi készült el és a tanulságokat ebből a sprintből) (Abrahamsson et. al. 2002).

A postgame fázisban zárul le a projekt, itt készül egy utolsó teljes teszt és a projekt dokumentációja. Itt már nem érkezhet új követelmény (Abrahamsson et. al. 2002).

A scrum keretrendszer meghatároz szerepeket is, amelyeket, mint már említettem a pregame fázisban osztanak ki. Ezek a szerepek a Scrum Master, Product owner és a Team (Szabó - Ribényi, 2018). A Scrum Master feladata a keretrendszer betartatása, valamit az akadályok elhárítása a csapat elől. A Product owner feladata a termékfejlesztés sikerének elérése, továbbá az ügyfél igényeinek felmérése (product backlog megírása és frissítése). A harmadik fontos szereplő a Team azaz a fejlesztő csapat, ők felelnek a tényleges fejlesztésért. A csapatszellem optimális kialakulásának érdekében ez a csapat álltalában öt-kilenc főből áll (Szabó - Ribényi, 2018). Abrahamsson et. al. (2002) említ még két fontos szereplőt, a megrendelőt, akinek feladata az új követelmények megfogalmazása, és a vezetőséget, akik meghozzák a végső döntéseket a követelmények között és a szerepek kiosztásában is.

A keretrendszer meghatároz különböző dokumentumokat is melyek közül a produkt backlogról már szó estett. Szabó és Ribényi (2018) további kettőt említ. A sprint backlog tartalmazza a produkt backlog elemeit, de már az elemek prioritásának megfelelő sorrendben. A harmadik dokumentum pedig a Burndown Chart amivel követni lehet a projekt haladását oly módon, hogy a grafikon függoleges tengelye a még elvégzendő feladatok számát mutatja a vízszintes pedig az eltelt időt.

Mint már említettem a scrum az egyik leghasználtabb agilis fejlesztési módszertan a Dima és Maassen (2018) és a Szabó és Ribenyi (2018) által végzett kutatásokon, ez jól mutatja a hatékonyságát, azonban a scrumnak is megvannak a hátrányai. Ezt a metódust csak kis csoportokra lehet alkalmazni (maximum tíz fő) (Abrahamsson et. al. 2002), mert nagyobb csoportoknál már nem jól alkalmazható a napi stand-up meeting valamit az álltalános csapat kommunikáció sem tud kialakulni.

# 3. Irodalomjegyzék

Abrahamsson, P. – Salo, O. – Ronkainen, J. – Warsta, J. (2002): Agile software development methods: Review and analysis, VTT publication 478, Espoo, Finland.

Chandra V. (2015): Comparison between Various Softver Development Methologies. Internacional Journal of Computer Applications, 2015, volume 131, no.9

Dima, A. M. - Maassen, M. A. (2018): From Waterfall to Agile Software: Development Models in the IT Sector, 2006 to 2018: Impacts on Company Management. Journal of International Studies, 2018, volume 11, issue 2, pp. 315-326

McConnell, S. (2004): Code Complete, második kiadás, Microsoft Press, USA

Szabó Bálint – Ribényi Máté (2018): Az agilis módszertanok megítélése a beosztottak és vezetők szemszögéből. Vezetéstudomány - Budapest Management Review, 49 (6). pp. 22-32.

Stober, T., & Hansmann, U. (2010). Agile Software Development: Best Practices for Large Software Development Projects. Berlin, Springer Verlag.