Budapesti Corvinus Egyetem

Gazdaságtudományi Kar

Számítástudományi Tanszék

**Forrás Hivatkozások számonkérésének automatizálása**

Készítette: Mátrai Tibor

Gazdaságinformatikus

2021

Burka Dávid

Tartalom

[1. Bevezetés 2](#_Toc85391975)

[2. Projektéletciklus modellek vizsgálata 3](#_Toc85391976)

[2.1 Vízesésmodell 3](#_Toc85391977)

[2.1.1 Követelmények 4](#_Toc85391978)

[2.1.2 Tervezés 4](#_Toc85391979)

[2.1.3 Implementáció/kódolás 5](#_Toc85391980)

[2.1.4 Tesztelés 5](#_Toc85391981)

[2.1.5 Támogatás és karbantartás 6](#_Toc85391982)

[2.1.6 A vízesés modell előnyei és hátrányai 6](#_Toc85391983)

[2.2 Agilis életciklus modellek 7](#_Toc85391984)

[2.2.1 Extreme programozás 7](#_Toc85391985)

[2.2.2 Scrum 9](#_Toc85391986)

[3. Keretrendszerek ismertetése 11](#_Toc85391987)

[3.1. Office Add-In 11](#_Toc85391988)

[3.1.1. Visual Studio Tools for Office (VSTO) 11](#_Toc85391989)

[3.1.2. Office Web Add-In 12](#_Toc85391990)

[3.2. Architektúra 15](#_Toc85391991)

[3.2.1. HTML és CSS 15](#_Toc85391992)

[3.2.2. JavaScript 15](#_Toc85391993)

[3.2.3 PHP 16](#_Toc85391994)

[4. Saját projekt bemutatása 16](#_Toc85391995)

[4.1 Követelmények 17](#_Toc85391996)

[5. Irodalomjegyzék 19](#_Toc85391997)

# 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

Ebben a fejezetben bemutatom milyen projekt életciklus modelleket lehetne használni egy az enyémhez hasonló fejlesztési projekt során, ha azt vállalati közegben valósítanak meg. Először a tradicionális vízesés modellt majd az újabb agilis modellek közül az Extreme programozást és a Scrumot mutatom be.

## 2.1 Vízesésmodell

1. ábra vízesésmodell (forrás: Chandra, 2015)

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.

### 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).

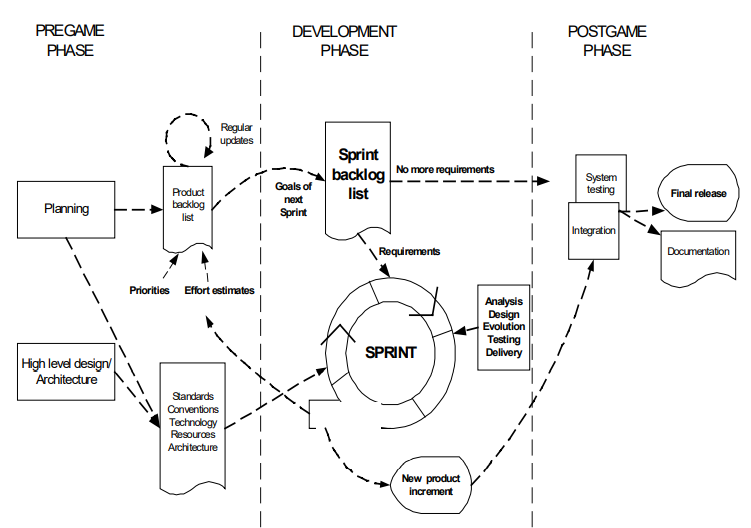
A következő fázis feladata az elkészült kiadás további tesztelése, hogy az tényleg átadhassuk a felhasználóknak. Itt is adhat hozzá még a megrendelő új követelmény ezeket már gyorsabban kell végre hajtani, mint az előzőket, ezek az igények a tervezés folyamata során készülnek bele a prioritás listába 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. Keretrendszerek ismertetése

Ebben a fejezetben bemutatom az automatizálás projekt közben használt vagy használható keretrendszereket. Szó lesz röviden a történetükről, működésükről, és a projekt szempontjából előnyeikről és hátrányaikról.

## 3.1. Office Add-In

Az Office Add-Inok a Microsoft cég által kínált megoldások olyan funkciók készítésére, amik alapból nem elérhető a Microsoft Office termékekben. A következőben két típusú Add-Int fogok bemutatni a VSTO-t és az újabb Office Web Add-Int. A projekt során ezek a rendszerek fogják képezni az alapját a kész terméknek, és segítségükkel ágyazom be az applikációt a Microsoft Word programba.

### 3.1.1. Visual Studio Tools for Office (VSTO)

Az első Visual Studio Tools for Office (VSTO) a 2003-as Microsoft Office verzióval együtt jelent meg, ez volt az első lehetőség arra, az Office applikáción kívül módosítani lehessen az Office fájlokat. Ez elött is volt lehetőség a fájlok módosítására kódolás segítségével, de akkor még csak makrók rögzítésére és írására volt lehetőség VBA segítségével, és ezeket nem lehetett álltalánosan használni, minden egyes új dokumentumnál az aktuális dokumentumra kellett őket testre szabni, ami sok munkaidőt igényelt alapból is, nem beszélve a hibák elhárításáról, amik fellépett minden egyes módosításkor. A VBA modell továbbá a védelem hiánya miatt a makró vírusok elterjedéséhez vezetett (Pirjan, 2015).

A 2003-as verzió azzal, hogy Visual Studioban lehetett használni, lehetővé tette a .NET keretrendszer használatát. Ezzel az újítással az applikációk a .NET assembly segítségével kapcsolódtak a Microsoft Office alkalmazásokhoz, amely védelmet jelentett az előbb említett vírusok ellen. Továbbá a Visual Studio és a .NET környezet lehetővé tette a C# kódolási nyelv használatát (Pirjan, 2015). Ez a verzió csak a Microsoft Office 2003 Word és Excel alkalmazásokat támogatta és azokban is csak dokumentum szintű változtatásokat tett lehetővé (Sempf - Jausovec, 2010).

2005-ben jelent meg a következő verzió VSTO 2005 SE néven. Ez a verzió már alap része volt a Visual Studio 2005 professional és magasabb verzióinak. Megjelent a többi 2003-as Office applikációk támogatása (Word, Excel, Power Point, Visio, Info-path, Outlook) és magasabb támogatást nyújtott a Windows Form controlok és a szerveroldali programozásnak. elkezdődött a 2007-es Office támogatás ez főleg csak a következő verzióban lett rendesen támogatva (Sempf - Jausovec, 2010).

A Visual Studio 2008-as professional verziójával jelent meg a harmadik verzió. Ez a kiadás lehetővé tette a COM (Component Object Model) Add-In-ok fejlesztését és ezzel lehetővé tette, hogy az Office applikációk szalagján (ribbon) jelenjenek meg a megírt funkciók, ezzel teljesen integrálni lehet az általunk megírt funkciókat az Office applikáció felhasználói felületébe (Pirjan, 2015).

A 2010-es Visual Studioval együtt jelent meg a negyedik verzió. Legnagyobb újítás a 64-bit-es rendszerek támogatása. Ebben a verzióban továbbá váltottak a Office client applikációk támogatásáról a SharePoint támogatásra, amivel lehetővé tették a SharePoint workflowk használatát a VSTO applikációkban (Sempf - Jausovec, 2010). Ennek egy újabb kiadása a jelenlegi verzió. Támogatást kapott a 2013-as Office applikációk többsége, valamit folyamatos frissítések érkeznek a .NET keretrendszerekre is ezáltal ez a verzió használható az össze 2013-asnál újabb Microsoft Office verzióra (Pirjan, 2015).

A jelenlegi verzió a Visual Studio 2019 verziójával fut 4.7.2-es .NET keretrendszert használ. Kompatibilis Windows 10-en futtatott 2013-as vagy újabb Microsoft Office applikációkkal (Excel, Word, InfoPath, Outlook, PowerPoint, Project, Visio), C# és VB programozási nyelveken.

A VSTO Add-Inoknak előnye a következőkben említett Web Ad-inokkal szemben, hogy lokálisan futnak, nincs szükség szerverre a használathoz, valamint a lokalitás és a .NET keretrendszernek köszönhetően elérik az operációs rendszeren keresztül a lokális fájlokat, fájlszerkezeteket. Jobban integrálódnak az Office által használt felhasználói felületbe, ezáltal a fejlesztésben nem részvevő felhasználó is könnyedén tudja használni hiszen hasonló ahhoz, amit már megszokott. Könnyebb fejleszteni hiszen csak C# vagy VB tudásra van szükség hozzá addig amig nem akarjuk szerverhez kötni (Raymond, 2017).

Fő hátránya a VSTO típusú kiegészítő alkalmazásoknak, hogy csak Windows operációs rendszeren lehet őket használni, pont az előbb említett .NET keretrendszer miatt. További hátrányuk Web add-inokkal szemben, hogy a közzétételt vagy értékesítést nem támogatja a Microsoft vagyis, ha értékesíteni szeretnénk a programunkat akkor arra saját megoldást kell találni. Ha csak megosztanánk akkor is saját rendszert kell rá tervezni, iroda környezetben például minden gépre fel kell telepíteni a rendszer gazdák álltal. (Raymond, 2017).

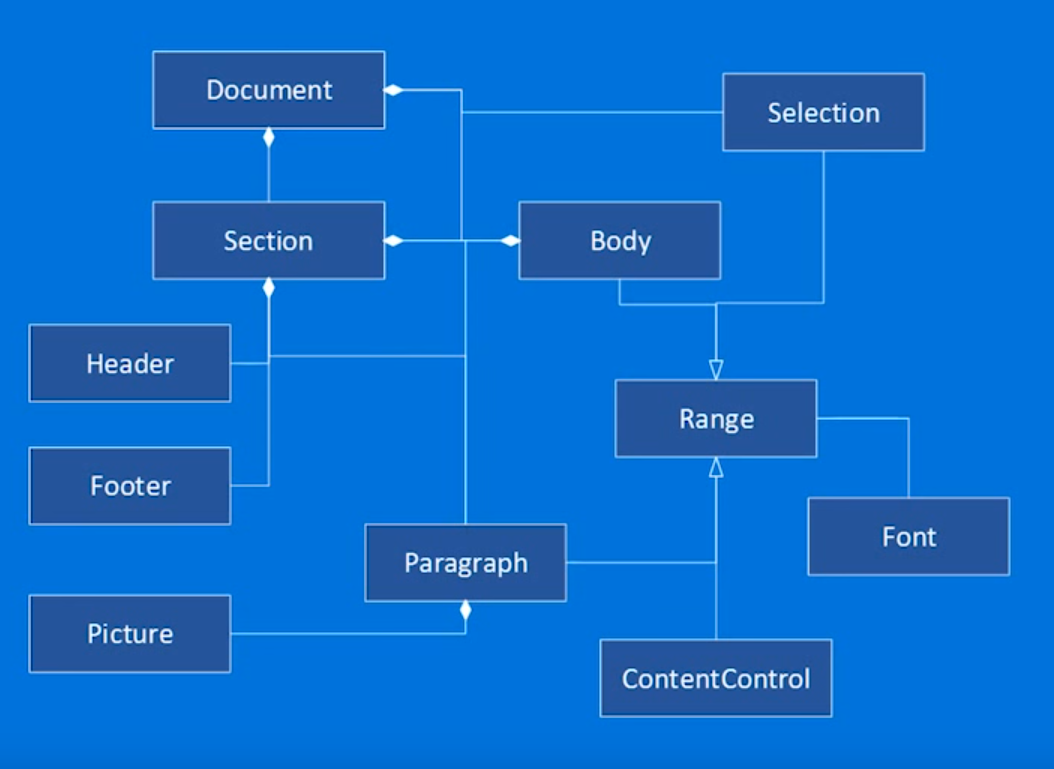
### 3.1.2. Office Web Add-In

Az Office Web Add-In technológia hasonlóan a VSTO-hoz lehetőséget ad a fejlesztőknek arra, hogy saját funkciókat adjanak a Microsoft Office termékekhez. Ennek a típusú add-innak az első alapjai a 2013-as Office verzióban és az akkor Visual Studioban jelentek. A fő megjelenésük viszont csak a 2016-os verzióban következett be mert ekkor kerültek bemutatásra az új fájl szerkezetek, amik közül a Wordhöz tartozó látható az 3. ábrán. Az új dokumentum szerkezet és a hozzá készült JavaScript API csomag segítségével lehetővé vált az Office web add-inokat fejlesztése.

A VSTO-kal ellentétben a Web Add-in-ok nem .NET és C# alapon futnak, a nevükből kitalálható, hogy web alapokon működnek. HTML és CSS adja a megjelenésnek az alapját és JavaScript a funkcionalitást. A JavaScript az a Microsoft által készített API és az említett dokumentum szerkezet segítségével tud módosításokat végezni a dokumentumokon (Balmori, J., 2015). Az API (Appication Programming Interfaces) definiciója: „egy olyan programozási interfészt, programozási felületet és annak részletes dokumentációját értjük, amelynek segítségével egy rendszer egy másik programhoz (esetleg rendszerprogramhoz) csatlakozhat. Ennek révén a másik programrendszer szolgáltatási használhatóak, anélkül, hogy a program belső részleteit ismerni kellene.” (industry4.hu).

A dokumentum szerkezet, az API és az API segítségével megírt JavaScript hármas adja az egyik felét egy Office Web Add-In-nak, a másik fele pedig egy XML file. Ez a fájl tartalmaz alap információkat mint, az Add-In neve, leírása, verziója, tartalmazza azt, hogy az add-in hogyan integrálódik az Office applikációba, melyik web szerveren található az add-in és tartalmazza a szükséges biztonsági feltételeket a használathoz (Microsoft, 2021).

A Web névből továbbá adódik, hogy az applikáció nem lokálisan, hanem valamilyen webszerveren fut. Ez lehet akármilyen alapú webszerver például .NET, vagy php (Microsoft, 2021).



3. ábra MS Office Word 2016 dokomentum szerkezet forrás: Balmori, J., (2015), 1:35

A Microsoft Word esetében a 3. ábrán látható az a szerkezet, amin keresztül az API tud kapcsolódni a dokumentumhoz. A 3. ábrán látható szerkezet alapján például, ha a dokumentumunkban egy kijelölt szöveget szeretnénk elérni akkor azt a Document->Selection keresztül tehetjük meg és egy Range elemet kapunk vissza, ami ahogy az ábrán is látható közvetlenül kapcsolódik a Font-hoz, szóval ezen keresztül végezhetünk változtatást a betűtípusban vagy egyéb megjelenében.

Az Office Web Add-In előnye a VSTO-val szemben az, hogy minden platformon müködik, lehet használni Windows-on, Androidon, iOS-en, és böngésző alkalmazásban is. Nem lokálisan fut tehát több gép is eléri egyszerre az adatbázist és nincs probléma a telepitéssel és a terjesztéssel. Céges környezetben például van lehetősége a rendszergazdának az összes felhasználónak integrálni a programot, többféle módon a saját irodájából. Továbbá a Microsoft Office-nak van saját online áruháza, ahonnan le lehet tölteni a már kész alkalmazásokat, meg lehet őket venni, és fel lehet rakni saját alkalmazást is. HTML-t és CSS-t használ a megjelenésre, tehát bármit meglehet valósítani felhasználói felületként (Balmori, 2015).

Hátránya a sok féle kódolási ismeret szükségessége, tudni kell HTML-t, CSS-t, JavaScriptet-t és azt keretrendszert is, amit a szerver használni fog. Az előbb előnyként említettem, hogy mindent meg lehet csinálni felhasználói felületként, viszont a Office applikációk szalagjába nem lehet elhelyezni az alkalmazást úgy, mint a VSTO-k esetében, tehát nem lesz természetes a kinézete a kész applikációnak (Balmori, 2015).

## 3.2. Architektúra

A következőkben a fejlesztési projektem során használt kódolási és programozási nyelveket fogom bemutatni. Szó lesz röviden a történetükről, és általános használatukról.

### 3.2.1. HTML és CSS

HTML (Hypertext Markup Language) az a nyelv, amivel megadjuk a szerkezetét a weboldalaknak, webalkalmazásoknak. Lehetőséget ad a fejlesztőnek, hogy létre hozzon dokumentumokat fejléccel, szöveggel, táblázatokkal, listákkal, képekkel és egyéb elemekkel (W3.org, 2021).

A HTML kód először 1990-ben jelent meg, egyike a World Wide Web három alap pillérének (HTML, URL, HTTP). Az alapok letétele után folyamatos fejlesztés jellemzi a verziókat. 1995-ben jelent meg a második verzió, 1997 elején a harmadik, 1997 decemberében pedig már a negyedik verzió volt a legfrissebb. A következő verzió, a HTML 5 2014-ben lett közzé téve a W3C által, mint a hivatalos, javasolt kódolási nyelv. Ennek a kiadásnak a legújabb W3C által javasolt verziója a HTML 5.2 (W3.org, 2017)

A HTML adja a szerkezetét a weboldalaknak, a CSS (Cascading Style Sheets) a kinézetét. „CSS aza nyelv, amivel megadjuk a kinézetét egy weboldalnak, beleértve a színeket, elrendezést és betűtípust. Lehetővé teszi a kinézet adaptálását más képernyő méretű ezközökre. A CSS független a HTML-től és használható bármilyen XML alapú markup nyelvvel. A HTML és CSS szétválasztása egyszerűsíti a weboldalak és webalkalmazások, karbantarását hiszen egy style sheettel változtathatjuk az összes oldalunkat.” (W3.org, 2021). Az előző idézetből egyértelműen következik, hogy a CSS egy külön álló kódolási nyelv, mégis elengedhetetlen része a weboldalaknak.

Mint már többször említettem a HTML és CSS nyelveket főleg weboldalak szerkezetének és kinézetének meghatározásához használják. Én a projektem során az Office Web Add-in felhasználói felületének kialakításához fogom használni.

### 3.2.2. JavaScript

A JavaScript programozási nyelv 1995-ben látta meg a napvilágot, Brendan Eich készítette a Netscape Communications céggel együtt működve (Netscape volt ebben az időben a legelterjedtebb böngésző). Azzal a céllal készült, hogy dinamikussá tudja tenni az eddig statikus weboldalakat kliens oldali programok segítségével (Ranjan et. al., 2020).

A legfontosabb történes a JavaScript korai életében ECMA standardizáció volt. Ez a folyamat 1996 novemberében kezdődött, ekkora már teljes használatban volt weboldalakon a programozási nyelv. Első standardizált verzió 1997-ben jelent meg, és a legutóbbi, a tízes verzió pedig 2019-ben jelent meg (Ranjan et. al., 2020).

A JavaScript nyelv elsősorban, a weboldalak kliens oldali dinamikájáért felel. Lehetséges vele módosítani a weboldalak statikus HTML és CSS szerkezetét. Különböző JavaScript alapú keretrendszerek segítségével lehetséges Back-End applikációkat is létrehozni. Én a projektem során az Office Web Add-In fejlesztésénél fogom használni, mint a kliens oldali logika.

### 3.2.3 PHP

PHP (Hypertext Preprocessor) egy elterjedt nyiltforráskódú álltalánosan használható programozási nyelv, ami kifejezetten jól használható webfejlesztéskor mert beágyazható HTML-be (php.net, 2021). 1994-ben jelent meg először, Rasmus Lerdorf készítette saját online önéletrajzának látogatottságának figyelésére, innen ered az eredeti név „Personal Home Page Tools”, 1995-ben készült el az első publikus verzió, ami más fejlesztők úgy használhattak ahogy szerették (php.net, 2021). Ezután 1997-ben jelent meg a harmadik majd 1999-ben a negyedik verzió, amely az új Zend Engine-nel működött. Jelenleg a hetedik verzió a legelterjedtebb, de már tesztelés alatt van a nyolcadik is (php.net, 2021).

PHP-t három fő területen használják. Szerver oldali applikációk írására, parancssori utasításokra operációs rendszereken és asztali alkalmazások fejlesztésére, bár ez nem javasolt az egyszerű grafikus felhasználói felület írási képesség hiánya miatt (php.net, 2021). Én a projektemben az Office Web Add-in szerver oldali alkalmazásának az megírásához fogom használni.

# Hivatkozási szabályok

Ebben a fejezetben bemutatom a Budapesti Corvinus egyetemen használt hivatkozási szabály rendszereket. Először általánosságban fogok beszélni a hivatkozások fontosságáról majd bemutatom a Harvard rendszer szabályait, amelyek a szakdolgozatom írásának kezdetén volt érvényben, majd az APA (American Psychological Association) stílusú hivatkozást, ami pedig most van érvényben az egyetemen. Ezeknek a szabály rendszereknek a bemutatása azért fontos, mert ezeknek a szabály rendszerek szabályainak ellenőrzésére szolgál az általam készített applikáció.

Miért kell hivatkozni

Harvard

APA

# 4. Hivatkozásszámonkérés automatizálás project bemutatása

A szakdolgozatom témájaként a Budapesti Corvinus Egyetem Gazdaságinformatikus BSc képzésének a szakszeminárium tárgy hivatkozásszámonkérés zárthelyi dolgozatának javításának az automatizálását választottam. A következőkben bemutatom a projektem lépéseit közben felmerült nehézségeket és az elkészült applikációt. Az első alfejezetben bemutatom a projekt kezdetekor fennálló megoldandó problémát.

## 4.1 A projekt által megoldandó probléma

A hivatkozásszámonkérés dolgozat automatizálásának az ötlete akkor született meg amikor először vettem részt a szakszeminárium 1 tárgyon. A tanárom panaszkodott, hogy nagyon sok idő ezeket a dolgozatokat javítani és ahhoz képest, hogy mennyi idő elmegy vele a tárgynak a jegyét nem befolyásolja akkor a mértékben. Viszont elhagyni sem lehet hiszen a tárgynak egyik kulcs feladata a helyes hivatkozás megtanítása. A javítási idő hosszának az oka a dolgozat szerkezete volt, egy dolgozatnak a javítása sok figyelmet igényelt. A dolgozat szerkezetét azonban nem akarták megváltoztatni, mert ez a módszer a gyakorlati tudást ellenőrizte.

A dolgozat egy másfél-két oldalas Microsoft Word dokumentum volt, amely egy tudományos szöveget tartalmazott tele hivatkozásokkal és a hozzátartozó irodalom jegyzékkel. Ezekben a hivatkozásokban és az irodalom jegyzékben 10 hivatkozási szabály hiba volt elrejtve, és a hallgatók feladata volt ezeket megtalálni és kijavítani.

Ez a szerkezet lehetővé tette, hogy az ellenőrizzék, hogy a hallgatók felismerik-e a helyes és helytelen hivatkozásokat és azokat tudják-e javítani, valamint egy olyan szituációt tudott utánozni, mint ami a szakdolgozat írás közben is felmerül, a saját dolgozat átnézését.

A javítás nehézsége pont ebből a szerkezetből adódott, hiszen ezeket a dolgozatokat javítani is úgy kell, mint ahogy megoldani, nagy figyelemmel végig nézni minden egyes hivatkozás és eldönteni, hogy helyes-e vagy sem. Ez a folyamat 5-10 percet vesz igénybe dolgozatonként, ami csoportonként 30 diákkal számolva körülbelül három és félórát vesz igénybe, és általában egy tanárnak nem csak egy csoportja van. A cél tehát ennek a javítási időnek a csökkentése, valamilyen automatizálás segítségével.

## 4.2 Automatizálási lehetőségek

Moodle

Office addin

Vsto

Web add-in

A kutatásaim alapján egy Word web add-in fejlesztése mellett döntöttem, a következőkben ezt a projektet fogom kifejteni. Először az általam összeírt felhasználó felületi és funkciós követelményeket írtam össze. A projekt github-ja megtalálható a <https://github.com/Matrait/Szakdolgozat> linken, a v.3.0 branch tartalmazza a legfrissebb verziót a hozzá tartozó hordozható devserverrel együtt.

## 4.1 Követelmények

Hivatkozás számonkérés

->kinézet

Két link egymás mellett gomb class-szal egymás mellett rendre Új feladat(ujfeladat.html), Javítás(javítás.html)

->funkciók

Uj feladat

->Kinézet

Szöveg doboz labellel „Kész feladat neve:”

kettő gomb rendre „új hiba”, „előző törlése”

link vissza a fő oldalra „vissza” class=”gomb”

->Funkciók

uj hiba felvétele

dokumentum neve szöveg dobozból változóba //ezt még át kell gondolni

mentés dok táblába

generált ID vissza kérése, változóba(dokID) másolás

kijelőlt szöveg beolvasása

Szöveg mentése változóba (szoveg)

kezdő pozíció mentése változóba (poz1)

vége pozíció mentése változóba (poz2)

felvett rekordok küldése szerverre

előző hiba törlése

feladat sor mentése

Gombnyomásra mentés másként ajánlott név: szöveg dobozból.

Javítás

->kinézet

egy legördülő lista

egy gomb „Betöltés”

link vissza a fő oldalra „vissza” class=”gomb”

->Funkciók

indulás

dok táblából dokumentum nevek beolvasása legördülő listába

Betöltés

legördulő listáből kiválasztott nevhez tartozó dokID lekérdezése dok táblából

szoveg táblából dokID-hez tartozó rekordok lekérése

foreach ciklussal poz1 és poz2 rekordok alapján pozíciók kiemelése

poz2 adat alapján szövegdobozok generálása eredeti szöveggel

# 5. Irodalomjegyzék

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

Balmori, J., (2015) „Word JavaScript API”. YouTube, 2015. 11. 18. <https://www.youtube.com/watch?v=EaLVfaj1mW8> Letöltés dátuma: 2021. 10. 16.

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

industry4.hu letöltés dátuma: 2021. 10. 16.

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

Microsoft Documentation (2021)<https://docs.microsoft.com/en-us/office/dev/add-ins/overview/office-add-ins#office-javascript-apis> letöltés dátuma: 2021. 10. 16.

php.net (2021) <https://www.php.net/manual/en/index.php> letöltés dátuma: 2021. 10. 17.

Pirjan, A. (2015): Solutions For Repurposing The Default Actions And States Of The Office Controls Through Component Object Model Add-Ins, Romanian Economic Business Review, Romanian-American University, vol. 9(1), pages 136-146, May.

Ranjan, A. – Sinha, A. – Battewad, R. (2020): JavaScript for modern Web Development, BPB Publications, New-Delhi

Raymond, B. (2017): “Comparing VSTO and Office Web add-ins,” TechGenix, 2017 május, <http://techgenix.com/comparing-vsto-and-office-web-add-ins-video/> Letöltés dátuma: 2021. 10. 16.

Sempf, B., - Jausovec, P. (2010): VSTO For Dummies. Hoboken, Wiley Publishing

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

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

W3.org (2017), HTML 5.2 is now a W3C Recommendation <https://www.w3.org/blog/news/archives/6696> letöltés dátuma: 2021.10.17.

W3.org (2021), <https://www.w3.org/standards/webdesign/htmlcss> letöltés dátuma: 2021. 10. 17.