Budapesti Corvinus Egyetem

Gazdaságtudományi Kar

Számítástudományi Tanszék

**Forrás Hivatkozások számonkérésének automatizálása**

Készítette: Mátrai Tibor

Gazdaságinformatikus

2021

Burka Dávid

Tartalom

[1. Bevezetés 2](#_Toc69342917)

[2. Projekéletciklus modellek vizsgálata 3](#_Toc69342918)

[2.1 Vízesésmodell 3](#_Toc69342919)

[2.1.1 Követelmények 4](#_Toc69342920)

[2.1.2 Tervezés 4](#_Toc69342921)

[2.1.3 Implementáció/kódolás 5](#_Toc69342922)

[2.1.4 Tesztelés 5](#_Toc69342923)

[2.1.5 Támogatás és karbantartás 6](#_Toc69342924)

[2.1.6 A vízesés modell előnyei és hátrányai 6](#_Toc69342925)

[2.2 Agilis életciklus modellek (1100) 6](#_Toc69342926)

# 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 hivatkozásában. 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 érdemes-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 három modellre az extrém programozási modellre, a tesztelésvezérelt modellre és végül 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.

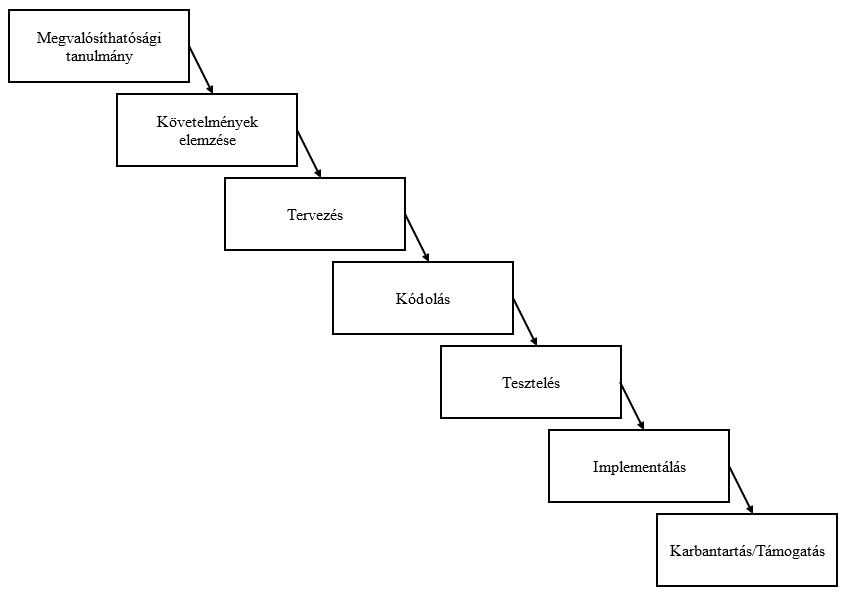
A dolgozat harmadik részében bemutatom a projektemet. Először leírom az oktatókkal közösen összeállított követelmény listát és megindokolom, miért ezt a számonkérési módszert választottam. Utána szó lesz a projekt közben alkalmazott életciklus modellről, a tervezés, a fejlesztés és a tesztelés lépéseiről. 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.

# 2. Projekéletciklus 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). 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 meg kell csinálni (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 majd 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 akkor bevezetni egy változtatást 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 (elő fordulhatnak átírások), 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. Amiennyiben 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 fokuszálhatnak 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 amig 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 tesztelők és a fejlesztők számának aránya egy régi kérdés, ha tradicionális projektről van szó, azaz nincs meg létező program, amit tovább fejlesztünk akkor körülbelül egy fejlesztőre jut egy tesztelő. Ha azonban már meglévő projektnek a tovább fejlesztéséről van szó akkor már több tesztelőre van szükség a regressziós tesztek miatt (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

## 2.2 Agilis életciklus modellek (1100)