ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Malaskova-Béres Krisztina**
2023	Hallgató törzskönyvi száma:	**T/002437/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Malaskova-Béres Krisztina**
Törzskönyvi száma: T/002437/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Vállalati rendszer tesztelésének kialakítása és automatizált tesztelés bevezetése
Improvement of the corporate system test capability and the introduction of test automation

Intézményi konzulens: Sziklai Zsolt
Külső konzulens:

Beadási határidő: 2022. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

A feladat

Tervezzen és készítsen egy olyan rendszert, ami egy adott vállalaton belül gördülékenyebbé teszi a tesztelést, és bevezeti a tesztautomatizálást. Az adott vállalat legyen légitársaság, ahol jelenleg a tesztelési folyamatok nincsenek megfelelően kiépítve, és a tesztelést manuálisan végzik.
A feladat, hogy a tesztelések elsődlegesen ne manuálisan legyenek végrehajtva, hanem a vállalat mérnökei által egy előzetes tesztelés keretein belül, illetve bizonyos rendszerek tesztelési folyamataira automatizált tesztek legyenek létrehozva úgy, hogy a megfelelő tesztelői dokumentációk, és adatok is rendelkezésre állnak.
Szükséges, hogy a tesztelési rendszerek szeparált környezetben fussanak, ahol más rendszerekkel nem interferálnak. A tesztelői környezetet úgy alakítsa ki, hogy ne történhessen más rendszerekkel olyan kölcsönhatás, ami miatt a tesztelés alkalmával valótlan eredmény születik.
Hozzon létre olyan folyamatot, ahol a mérnökök végső tesztelést képesek elvégezni az ügyfelekkel közösen, valamint adjon megoldást arra, hogy a teszt rendszert hogyan tartja naprakész állapotban.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- a lehetséges megvalósítási módokat és megoldásokat,
- a részletes rendszertervet és annak indoklását,
- a megvalósítás során használt technológiák bemutatását,
- a megvalósítás menetét, a munkafázisok és a tapasztalatok leírását,
- a tesztelési módszereket, tesztelési terveket és a tesztelés eredményeit,
- az elkészült rendszer alkalmazási, és továbbfejlesztési lehetőségeit,
- online a kifejlesztett rendszer forráskódját, illetve a készített dokumentációkat.

ÓBUDAI EGYETEM Neumann János Informatikai Kar Budapest 1 PH.

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

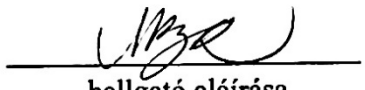
.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022 december 13



hallgató aláírása

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
Malaskova-Béres Krisztina	[redacted]	Esti
Telefon:	Levelezési cím (pl: lakcím):	
[redacted]		

Szakdolgozat / Diplomamunka[1] címe magyarul:
Vállalati rendszer tesztelésének kialakítása és automatizált tesztelés bevezetése

Szakdolgozat / Diplomamunka[2] címe angolul:
Improvement of the corporate system test capability and the introduction of test automation

Intézményi konzulens:
Sziklai Zsolt

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.11	Információk ismertetése, témák átbeszélse	
2.	2022.05.03	Eddigi munka áttekintése, javítások átbeszélése	
3.	2022.05.10	Javítások átbeszélése, tivábbi kérdések tisztázása	
4.	2022.05.13	Véglegesítés, utolsó javítások átbeszélése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022. május 13.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun kód:	Tagozat:
Malaskova-Béres Krisztina	[redacted]	Esti
Telefon:	Levelezési cím (pl: lakcím):	
[redacted]		

Szakdolgozat / Diplomamunka[5] címe magyarul:
Vállalati rendszer tesztelésének kialakítása és automatizált tesztelés bevezetése

Szakdolgozat / Diplomamunka[6] címe angolul:
Improvement of the corporate system test capability and the introduction of test automation

Intézményi konzulens:
Sziklai Zsolt

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.11.04	Eddigi munka áttekintése, javítások átbeszélése	Sziklai
2.	2022.12.01	Javítások és eddigi munka átbeszélése, javítási javaslatok	Sziklai
3.	2022.12.08	Javítások átbeszélése, kiegészítések megírásához szükséges információk	Sziklai
4.	2022.12.12	Véglegesítés, utolsó javítások átbeszélése	Sziklai

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[7] tantárgy követelményét teljesítette, beszámolóra / védésre[8] bocsátható.

Budapest, 2022. december 12.

Sziklai

intézményi konzulens

[5] Megfelelő aláhúzandó!
[6] Megfelelő aláhúzandó!
[7] Megfelelő aláhúzandó!
[8] Megfelelő aláhúzandó!

Tartalomjegyzék

Bevezetés

Napjainkban kezd egyre keresettebb lenni a cégek körében a szoftverek átfogóbb tesztelése, és ezen tesztek automatizálása. Tizenegy évnyi szakmai tapasztalatom alatt rengetegszer belefutottam abba, hogy a vállalatok nem hoznak létre elég erőforrást, hogy a termékük megfelelően le legyen tesztelve, és hogy hibamentesen kerüljön az ügyfélhez, továbbá nem építik bele a fejlesztési folyamataikba a tesztelést.

Munkám során én is sokat szenvedtem azzal, hogy nem kaptam elég támogatást a szoftverek tesztelésével kapcsolatban, és hogy a tesztelésnek teret tudjak teremteni egy adott vállalaton belül, ezért elhatároztam, hogy keresni fogok egy olyan vállalatot, ahol magát a tesztelést fel tudom építeni az alapoktól, és a tesztek automatizálását be tudom vezetni.

Ezután nem sokkal volt szerencsém egy olyan céghez felvételt nyerni, ahol azt a feladatot kaptam, hogy az üzlettől át kell vennem a tesztelést, a fejlesztési folyamatokba be kell építenem a tesztelési folyamatokat, és a manuális tesztelés mellett be kell vezetnem a tesztautomatizálást is. A módszerek, és alkalmazások tekintetében szabad kezet kaptam.

A termékek egy multinacionális informatikai területtel is rendelkező cég rendszeréhez kapcsolódó szoftverek, melyek magukba foglalják a dokumentációs eszközöket, és a könyvelési szoftvereket is mint pl.: számlázás, szerződés, és ezen folyamatok, és dokumentumok állapotának nyomonkövetése. Kihívása a rendszer átformálásának, hogy a termékeket nem csak cégen belül alkalmazzák, hanem cégen kívüli ügyfelek is dolgoznak vele, ezért a hibamentesség hatványozottan fontos.

A feladat megoldása során további átalakítást igénylő területekkel szembesültem, mint például a tesztkörnyezetek átalakítása, illetve azok egymástól való szeparációja. Megoldásképpen úgy döntöttem, hogy olyan tesztkörnyezetet szükséges kiépíteni, ahol valós leképzett adatokkal tudnak a tesztelők tesztelni, illetve az adatbázisokban tárolt adatokat manipulálni, anélkül, hogy eltérés lenne a valós rendszer, és a tesztrendszer között, továbbá ahol a tesztrendszerek közötti anomáliákat ki lehet küszöbölni.

A tervezés alkalmával több megoldást is megvizsgáltam, és figyelembe vettem. Szempontjaim között volt a megvalósítási idő hossza, az erőforrások kiszámítása, illetve a költségek kalkulációja melyek a fejlesztéssel, és folyamat építéssel járnak. A folyamat fejlesztés alapjait az ISTQB [1], és ASPICE [2] módszertanokból használtam fel, mivel a tesztelési iparágban ez bizonyult a legalaposabbnak, és leginkább kidolgozottnak. Továbbá ezt a két módszertant használja az autóipar is, ahol a tesztelésnek kiemelkedő szerepe van, és szigorú szabályok mentén képesek olyan tesztelési megoldásokat eszközölni, amelyek egy ellenőrzött rendszerben képesek a lehető legminimálisabbra redukálni a szoftverekben előforduló hibákat.
Az adott vállalatnál több rendszer is használatban van, azonban feladatom során csak három rendszeren szükséges elvégeznem a fejlesztéseket. A három alkalmazásból kettő

szorosan összekapcsolódik, és egy teljesen független, ezért a feladatot két fő részre fogom osztani.

Első részben az összevont két alkalmazást ismertetem. Ezek számlázó, illetve főkönyvi rendszerek. Második részben pedig egy dokumentum formázó alkalmazást mutatok be, ahol az ipari elvárásoknak megfelelően van lehetőség automatizálva formázni egy Word [3] dokumentumot.

Célom, hogy egy olyan komplex megoldást valósítsak meg, ahol a tesztelési folyamatok automatizáltak, gördülékenyek, és a kiadott szoftver, mint az üzlet, mint az ügyfél számára felhasználóbarát és hibamentes.

1. Aktuális könyvelői rendszerek ismertetése

1.1 Számlázó rendszer

A Számlázó rendszer a Főkönyvi rendszer egyik alrendszere, ahol lehetőség van kezelni a bejövő számlákat, és szerződéseket, illetve rögzíteni azokat, kiválasztva, hogy melyik általános könyvelési rendszeren keresztül kerüljenek be a főkönyvelésbe.

Ez a rendszer öt fő modulból tevődik össze:

- Szerződések modul: A szerződésekben szereplő adatok az irányító modulban kerülnek felhasználásra.
- Számlák modul: Ebben a modulban rögzítik a számlákat, amiket jóvá kell hagyni. A jóváhagyást követően kerülnek a Főkönyvi rendszerbe.
- Irányító modul: Biztosítja, hogy a megfelelő költségek a megfelelő szakaszban jelenjenek meg, majd ezekből elemzés készíthető, amely felhasználható egyfajta fizetési előrejelzésként a Jelentés modulban.
- Törzsadat modul: A törzsadatok a szerződésekhez és számlákhoz felhasznált összes adatot tárolják, mint például a pénznemet.
- Jelentés modul: Megtekinthetőek a költségek különböző részletei egy jelentés keretein belül.

Az öt fő modul meghatározott függőségek alapján kapcsolódik egymáshoz.

1.2 Főkönyvi rendszer

A Főkönyvi rendszerben a számlák három különböző folyamaton mehetnek keresztül, ahol a bejövő, és kimenő fizetéseket kezelik. Ezen belül három folyamatot különböztetünk meg:

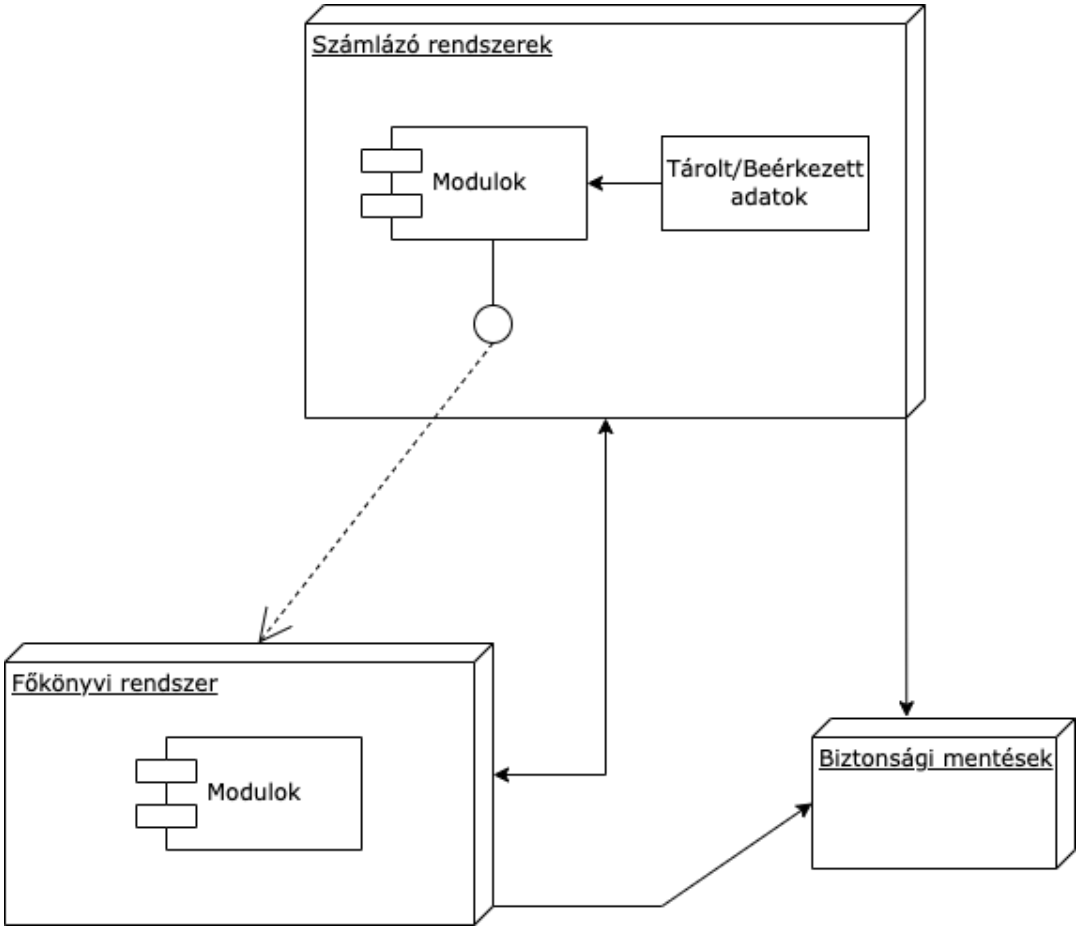
- Fizetendő Számlák
- Követelések
- Kiküldetések regisztrációja

Minden folyamathoz más jogosultság társul, ami azt jelenti, hogy minden állapotban van egy adott felhasználó, akinek az adott állapotban lévő dokumentum tovább léptetésére van jogosultsága. Ez az információ a teszt környezet kapcsán lesz majd fontos.

1.3 A két rendszer magas szintű együttműködése

A Számlázó rendszerben létre jön egy számla, amit az adatok ellenőrzése után a megfelelő modulhoz társítanak. A társítás függ a számla típusától, illetve egyéb adataitól. A modulhoz való társítást követően a számlát tovább küldik a Főkönyvi rendszerbe, ahol kap egy azonosítót. Az azonosítást követően a számla visszakerül a Számlázó rendszerbe, ahol megvizsgálják, hogy a számla kiállítójának van-e tartozás. A tartozás mértékétől

függően a különbözeti összeg eltárolásra kerül, és a számlához hozzáfűzik. Ezután a számlát visszaküldik a Főkönyvi rendszerben, ahol a kifizetés történik. Az általam draw.io-ban [5] elkészített [*1.1* ábra] szemlélteti a két rendszer kapcsolatát.



1.1 ábra: Számlázó renszer, és Főkönyvi rendszer kapcsolata

A kifizetést követően a Számla véglegesített stádiumba kerül, ahol már semmilyen módosítást nem lehet rajta végrehajtani. Mint az látható is, a két rendszer között folyamatos a kommunikáció.

1.4 A rendszerhez felépített tesztrendszer

A fentebb ismertetett két rendszerhez egy tesztrendszer lett létrehozva, ami összesen öt teszt környezetet tartalmaz.

Ezek a teszt környezetek az alábbiak szerint vannak jelenleg felhasználva:

- Fejlesztői környezet: Ahol a fejlesztők fejlesztik az adott két rendszert.

- Tesztelői környezet: Ahol a fejlesztők és üzleti elemzők tesztelik a javításokat, és fejlesztéseket. Jelenleg a tesztelés szuper adminisztrátori jogokkal történik.
- Üzleti környezet 1: Ahol az üzlet végzi az ad-hoc, és kisebb teszteléseket.
- Üzleti környezet 2: Ahol az üzlet végzi a release teszteléseket.
- Üres környezet: Jelenleg kihasználatlan.

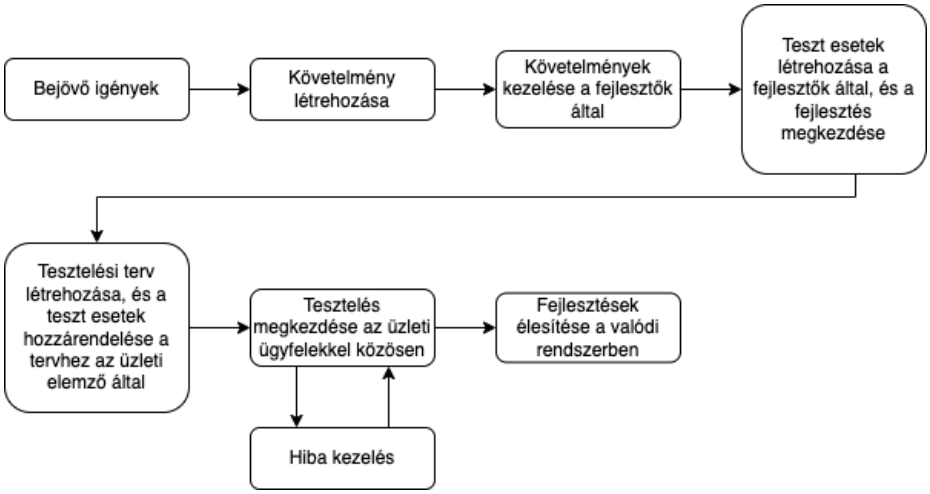
A felsorolt teszt környezetek nem szeparált környezetek, ezért vannak esetek, mikor a tesztelői környezet Számlázó rendszere van összekötve az üzleti teszt környezet Főkönyvi rendszerével. Ezen kívül egyéb összekötési variációk is létrejöhetnek.

Hátránya, hogy így az adatok nem minden esetben kerülnek át egyik rendszerből a másikba, vagy előfordulhat az is, hogy az adatok torzulnak a két rendszer kommunikációja közben, esetleg olyan számlák, vagy szerződések kerülnek a Főkönyvi rendszerbe, vagy a Főkönyvi rendszerből a Számlázó rendszerbe, amik már véglegesítettek, vagy nem relevánsak. Tehát régebbi adatok kerültek újbóli feldolgozásra.

Az általam draw.io-ban [5] elkészített [*1.1* ábra] mutatja, hogy a biztonsági mentések a teszt környezet részét képezik. A biztonsági mentések egy része automatikusan történik, míg más részei manuálisan történnek a IT Support csapat közreműködésével.

1.5 A rendszerhez tartozó tesztelési munkafolyamatok

A jelenlegi folyamatban nincsenek teszt mérnökeink, akik felelősek lennének a termék teszteléséért, a teszt esetek létrehozásáért, és a hibák jelentéséért. A tesztelői folyamatokat a fejlesztők, üzleti elemzők, és üzleti ügyfelek végzik. Nem tartunk átfogó belső tesztelési tevékenységeket, mielőtt a termék az üzleti ügyfelekhez kerülne.



1.2 ábra: A tesztelői és fejlesztői munkafolyamat ábrája

A jelenlegi munkafolyamat a következőképpen épül fel, amit a draw-io-ban [5] készítettem el [
1.2 ábra]:

- Beérkeznek az üzleti igények, és ezt követelménybe foglalja az üzleti elemző.
- A követelmények fejlesztését megkezdik a fejlesztők.
- A fejlesztők a második lépéssel párhuzamosan megkezdik a teszt esetek megírását.
- Az üzleti elemző elkészíti a tesztelési tervet, amelyekhez hozzá adja a teszt eseteket, és kijelöli az adott teszt esetekhez tartozó tesztelőket az üzleti ügyfelek részéről.
- Amennyiben az üzleti elemző hibákat talál az üzleti ügyféllel való közös tesztelés alkalmával, úgy azokat dokumentálja, és hozzá csatolja a releváns követelményekhez.

2. Problémák definiálása a jelenlegi működés kapcsán

A munkafolyamatban, és a teszt környezet felépítésében több probléma is jelen van, ha az ISTQB szabványokat vesszük figyelembe. Ilyen probléma például az, hogy nincsen dedikált tesztelő az adott rendszer teszteléséhez, illetve a tesztelési folyamatokat olyan pozícióban lévő emberek végzik, akiknek nem ez a feladatuk. A továbbiakban ezen problémákat összegyűjtöttem, és ismertetem.

2.1 Problémák a munkafolyamatban

A mukafolyamat áttekintését követően összegyűjtöttem hat problémát, melyekre megoldásokat szeretnék alkalmazni.

Ezek a problémák a következőek:

- A fejlesztők írják meg a teszt esteket, és az üzleti elemző át kell hogy szerkessze ezeket a teszt esteket, hogy az üzleti ügyfelek számára érthető legyen.
- Bizonyos esetekben előfordul, hogy nincs elég idő, és erőforrás a hibák nyomonkövetésére.
- Nincsen külön hibajegy a felfedezett hibákról. A tesztelői dokumentáció tartalmazza a felfedezett hibákat, és az új fejlesztéseket is.
- Nincsen megelőző tesztelés mielőtt az üzleti ügyfelek hozzá látnának a teszteléshez.
- Nincsen dedikált tesztelő, aki elvégezné a tesztelői feladatokat, mint például a teszt esetek létrehozása, a hibák dokumentálása, a teszt esetek hozzáfűzése a tesztelési tervhez, a tesztelési jegyzőkönyv elkészítése, illetve az ügyféllel való közös tesztelés lebonyolítása.
- Az üzleti elemző állítja össze a tesztelési tervet.

2.2 Problémák a tesztrendszerben

Az aktuális teszt környezetek egy nagy komplexitással rendelkező rendszer, ami további hét alkalmazáshoz kapcsolódik. Ezen rendszerek nem képezik a jelen dokumentum részét.

A rendszerek elemei keresztbe vannak kötve egymással, és ez súlyos anomáliákat képes generálni a tesztelési tevékenység folyamán. Ilyen probléma például az alkalmazás UI felületén lévő információk valótlansága, vagy azok elérhetetlensége a Számlázó rendszer használata közben, mivel nem a megfelelő Főkönyvi rendszerrel lett összekötve a teszt környezet a tesztelés előtti felkészülés folyamán.

2.3 Hiányzó tesztelői dokumentumok

A teszteléshez elengedhetetlen a megfelelő dokumentumok elkészítése, azonban a tesztelési terven, a teszt eseteken, és a hibajegyeken kívül nincsen dokumentum a tesztelésről, és az ahhoz kapcsolódó tevékenységekről. A hiányzó dokumentumok felsorolásához az ISTQB CTFL [4] tananyagban szereplő dokumentumokat vettem alapul, melyek az alábbiak:

- Teszt Stratégia: Magas szintű dokumentum ami definiálja a tesztelési szinteket, és típusokat, hogy a termék tesztelése megvalósítható legyen.
- Tesztelési irányelv: Magas szintű dokumentum ami definiálja a tesztelési elveket, módszereket, és minden fontos tesztelési célt az adott organizációban.
- Követelmények nyomonkövethetőségi mátrixa: Megmutatja a követelmények, és teszt esetek kapcsolatát, illetve a teszt lefedettséget.
- Hiba jelentés: Tartalmazza az adott tesztelés folyamán előforduló összes hibát, azok fontossági szintjeit, és állapotát.
- Tesztelést összegző jelentés: Magas szintű dokumentum, amely összegzi a tesztelés folyamán elvégzett tevékenységeket, és eredményeket.
- Tesztelési terv: Magas szintű dokumentum ami definiálja a tesztelés mérföldköveit, erőforrásait, időbeosztását, és technikai részleteit. Részét képezi a teszt stratégia.

3. Megoldási javaslatok

A megoldási javaslatok összegyűjtése során igyekeztem minden felsorolt problémára megoldást találni, és a lehetséges opciókat összehasonlítani, ahol mérlegeltem, melyik megvalosítás lenne optimális költség, idő és erőforrás szempontjából. Ezeket a megoldásokat az alfejezetekben ismertetem.

3.1 Megoldási javaslatok a munkafolyamatban

A munkafolyamatnál figyelembe vettem, hogy jelenleg milyen erőforrással rendelkezünk a vállalatnál, és hogy van-e lehetőség erőforrás allokációra. Tapasztalataim alapján radikális változást nem célszerű alkalmazni, mert a jelenleg megszokott munkafolyamatokból való kilépés egy új folyamatba lassíthatja a munkát. Ez idő, és költség kiesést jelent. Törekedtem a minimális változtatásra, ami az eredmény tekintetében szignifikáns javulást idéz elő.

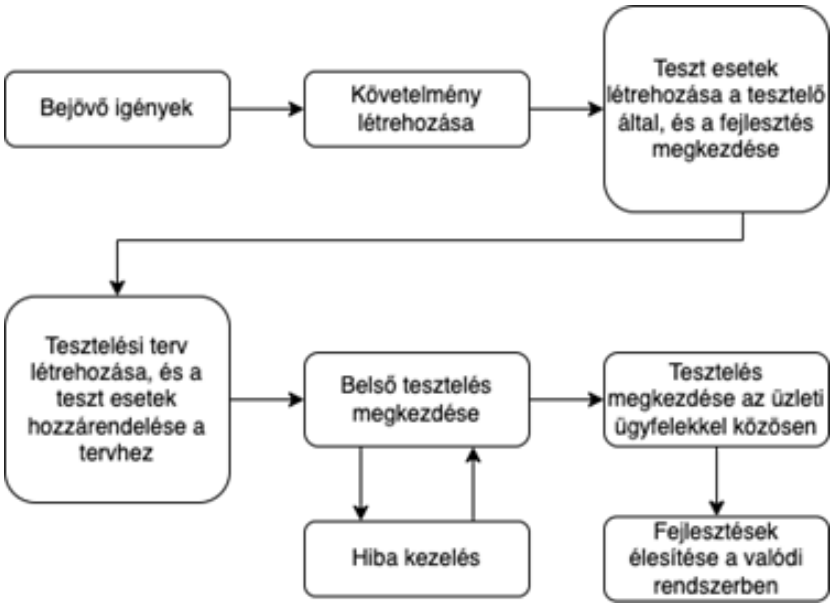
3.1.1 Első megoldási javaslat

Szükséges legalább egy tesztelő alkalmazása a projekthez. Munkája gyorsítaná az effektív munkavégzés idejét, hiszen az üzleti elemző, és fejlesztők, a tesztelési feladatok elvégzése nélkül több erőforrást tudnak allokálni a még hátralévő feladatok elvégzésére.

Tesztelői feladatok tekintetében is előnyt jelentene, hiszen a tesztelő által elkészített dokumentumok pontosabban tudnának rámutatni a tesztelés céljára, és a kritikus tesztelői szemléletet alkalmazva a hiba detekció is növekedést mutathat, ami a rendszer stabilitását, és hibamentességét segítené elő.

Fontos szempont még, hogy a tesztelésekbe továbbra is be legyenek vonva az üzleti ügyfelek, és a tesztelő segítségével le legyenek vezényelve az ügyféllel történő közös tesztelési alkalmak, így növelve az ügyfél bizalmát. További előnye ennek a javaslatnak, hogy lehetőség van egy előzetes tesztelésre a tesztelő által, még az üzleti ügyfelekkel történő közös tesztelés megkezdése előtt, így a már korábban detektált hibákat hamarabb van lehetőség letesztelni, és az esetleges új hibákat még az üzleti tesztelés előtt lehetőség van javítani.

A tesztelői dokumentumok fejlesztése már a követelmény megírását követően egyből elkezdhetőek, így nem a fejlesztési időből kell időt allokálni rá. Ilyen például a teszt esetek megírása. A folyamatot a draw.io-ban [5] általam elkészített [*3.1* ábra] mutatja be:



3.1 ábra: A tesztelői és fejlesztői munkafolyamat első módosítási javaslata

Lehetőség van a tesztelés automatizálásra is, így több szintű tesztelést lehet elvégezni. Ilyen szintek például az integrációs tesztelés, system testing (rendszer tesztelés), UAT (User Acceptance Testing – Felhasználó Elfogadásii Tesztelés), és az ezeken belüli funkcionális tesztelés, és nem funkcionális tesztelés. Az említett tesztelési szinteket több különböző módon lehet csoportosítani. Ilyen csoportosítások lehetnek a következőek:

- Regresszió tesztelés,
- Stressz tesztelés,
- Load tesztelés,
- Error Handling – Hibakezelés,
- Exploratory tesztelés – Felfedező tesztelés.[1]

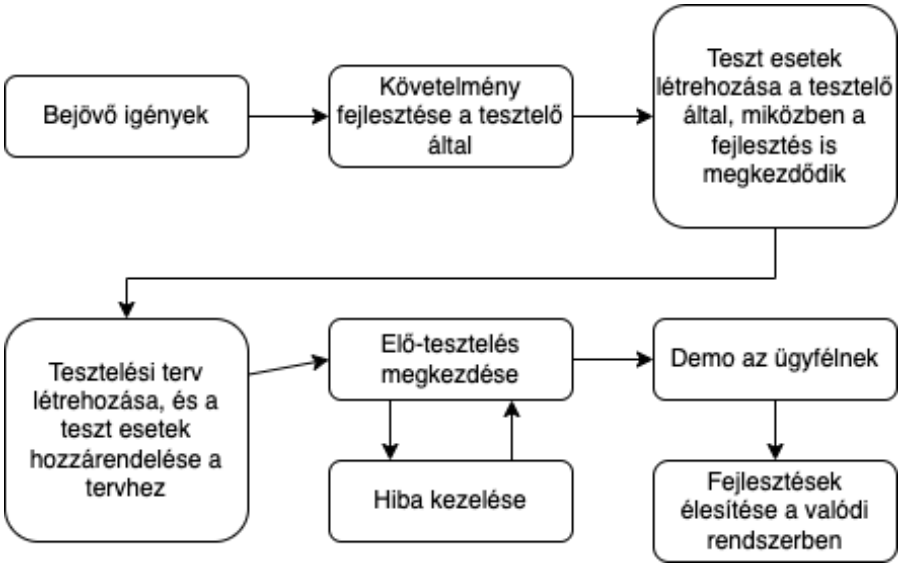
3.1.2 Második megoldási javaslat

Ebben a javaslatban is fontos szerepet játszik a tesztelő, de itt a követelmények megírásában is részt venne. A követelmények megírásával párhuzamosan, minden megírt követelményhez elkészítené a teszt eseteket, így biztosítva az azonnali lefedettséget.

A teszt esetek megírását követően, mikor a fejlesztők is elvégezték a fejlesztést, a tesztelő megkezdené az elő-tesztelést. Amennyiben hibát fedez fel a rendszerben, úgy azt dokumentálná, és ezt követően részt venne a hiba nyomonkövetésében is.

Ezzel párhuzamosan feljesztené a további tesztelői dokumentumokat visszamenőlegesen, és fejlesztené a automatizált teszteket is.

Az üzleti ügyfelekkel már nem venne részt közös tesztelésen, ellenben az üzleti elemző a tesztelővel közösen tartana demo-t az üzleti ügyfelek számára. Ennek folyamatát a draw.io-ban [5] általam elkészített [*3.2* ábra] mutatja be:



3.2 ábra: A tesztelői és fejlesztői munkafolyamat második módosítási javaslata

3.1.3 A két megoldás összehasonlítása

Tekintve a két esetet az elsőt tartom optimális megoldásnak. Indoklásom, hogy a második esetben a tesztelő által megírt követelmények nem feltétlenül kerülnek review-zásra mások által, és így a követelmény szintjén rizikónak van kitéve a szoftver hibamentessége, mert maradhat hiba a követelmények definiálásában. Ezzel szemben az első esetben az üzleti elemző által megírt követelmények a tesztelő teszt eset fejlesztési tevékenysége közben review-zva vannak, és így a hiba lehetőség redukálható.

Figyelembe kell venni azt is, hogy az üzleti ügyfél előnyben részesíti a tesztelői meglátást, és nem feltétlenül szeretne kimaradni a tesztelésből. Fontos számukra, hogy az éles rendszerre kikerült fejlesztéseket ők is kipróbálják a release előtt, és egyfajta biztonságérzetet nyújt a számukra, ha közben a fejlesztői csapat tagjai is jelen vannak. Ennek az oka, hogy az esetleges problémák, és hibák jelentkezésekor azonnali visszajelzést tudnak adni a fejlesztői csapatnak, és elakadás esetén segítségnyújtásban részesülnek. Abban az esetben, ha ezek a problémák a fejlesztői csapat jelenléte nélkül jelentkeznek, az megakaszthatja az üzleti ügyfelek munkáját, és ez időbeni kiesést jelent a vállalat számára is, hiszen nem minden esetben megoldható a hiba azonnali javítása az éles rendszeren. Emiatt a javítás drágább is lehet.

A felsorolt érvek tekintetében az első megoldási javaslatot fogom választani.

3.2 Megoldási javaslatok a tesztrendszerre

A teszt környezetek jelenlegi működése anomáliákat okoz tesztelés közben, és emiatt hamis eredményeket kapnak azok, akik a tesztelést végzik. A megoldási javaslatok tervezése alatt nehézséget okozott, hogy a kiemelt két rendszer egyik eleme további hét másik rendszerhez is kapcsolódik.

3.2.1 Első megoldási javaslat

A teszt környezetek dependenciáját olyan módon kell kialakítani, hogy az bármelyik másik rendszerhez kapcsolódni tudjon oly módon, hogy az a tesztelés alatt nem okoz anomáliákat. Erre megoldás lehet egy hibakód bele integrálása a szoftverbe, mely figyelmezteti a felhasználót, hogy bizonyos adatok nem fognak a tesztelés folyamán rendelkezésre állni. Ha ezt a problémát a felhasználó ki szeretné javítani, akkor lehetősége van az IT Support csapatnak jeleznie, akik elvégzik a megfelelő adat integrációt, vagy átkötik az aktuális rendszert egy másik, aktuális adatot használó rendszerhez.

A megoldási javaslat alapja a jelenlegi rendszer működésének megtartása a megfelelő hiba kezelésének módszerével.
Hátránya, hogy a hiba jelzés alkalmával a felhasználó nem tudja folytatni a tesztelést az adott rendszeren.

3.2.2 Második megoldási javaslat

A tesztrendszeren belüli teszt környezetek egymástól való elkülönítése megoldás lehet a tesztelés közben felmerülő anomáliákra. Ezzel a módszerrel nincs lehetőség a rendszereket keresztbe kötni.

A megoldás az lenne, hogy a tesztrendszeren belül a jelenlegi teszt környezeteket elzárjuk egymástól, így az átkötés megvalósíthatatlan lesz. Figyelni kell arra is, hogy a két rendszer felülírása azonos időben történjen az éles rendszerről, mert ha ez a feltétel nem valósul meg, akkor újfent hamis adatokat kaphatunk tesztelés közben. Ennek felelőssége az IT Support csapatra hárul. Megoldást jelent egy olyan script létrehozása, ami azonos időben indítja el a két rendszer mentését az éles környezetből a teszt környezetbe, így a hiba jelentkezése nem fog fent állni.

3.2.3 A két megoldás összehasonlítása

A második megoldás tesztelési szempontból optimális, mivel nem akasztja meg a tesztelést végző személyt, és nem okoz folyamtos IT Support támogatást. A felmerült problémára, hogy a két rendszer mentése azonos időben kell történjen, egy script nyújthat

segítséget, így részben elindul a tesztrendszerek létrehozásának, és frissítésének automatizációja.

Az első megoldásban nagy a rizikó, mert vannak esetek mikor a tesztelés szűkös határidőhöz kötött, és az IT Support sem elérhető. További szempontjaim között volt, hogy a teszt környezetek közötti keresztbe kötést szeretném megszüntetni, mert előfordulhat, hogy egyedül marad valamelyik rendszer, és nem kötődik hozzá semmilyen másik rendszer. Ha ilyen esetben ellenőrzés nélkül valaki elkezdi használni a fő, vagy alrendszert, és ebből von le konzekvenciát, az hamis hiba jelenségeket okozhat. Ezeknek a hamis hibajelenségeknek a root cause analízise időigényes, és fölösleges idő kiesés.
Emiatt az első megoldást választom.

4. Könyvelői rendszer fejlesztési tervei

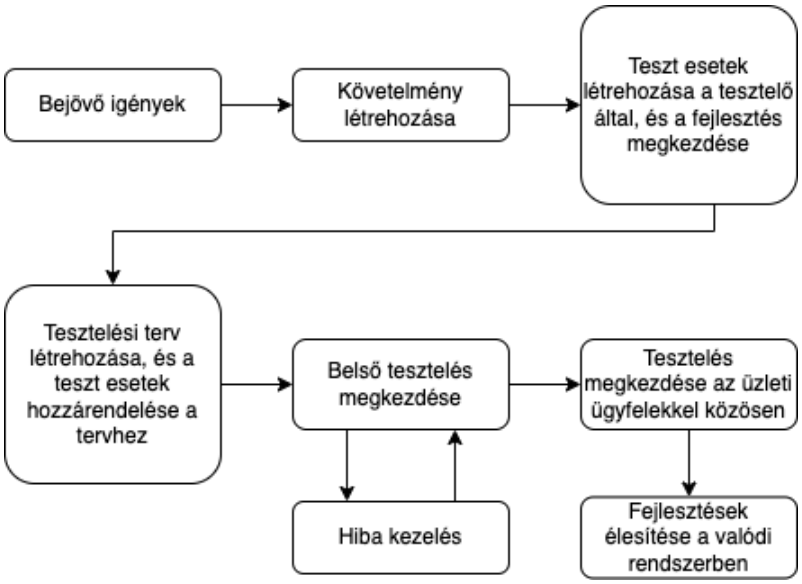
4.1 Munkafolyamatok fejlesztési terve

Az általam elkészített új folyamat terve szerint egy további tevékenységgel egészül ki a tesztelési munkafolyamat. Illetve új erőforrást is igénybe kell venni, hogy a fejlesztés megvalósulhasson.

Ahhoz, hogy a tesztelés gördülékenyen tudjon működni, legalább egy tesztelőt alkalmazni kell a projekten. A tesztelő feladatai közé tartozik, hogy elkészíti a tesztelési terv egy részét, közösen a Teszt Menedzserrel, létrehozza a teszt eseteket, és megírja a szükséges tesztelői dokumentumokat, mint például a hiba jelentést, és a tesztelési jegyzőkönyvet, amiben szerepelnek a tesztelési eredmények.

Az új folyamat a következőképpen alakul amit az általam draw.io-ban [5] elkészített [*4.1* ábra] is szemléltet:

- Beérkeznek a fejlesztési igények az üzleti ügyfelek részéről.
- Az üzleti elemző elkészíti a követelmények leírását.
- A követelmény specifikáció alapján a tesztelő megírja a követelményekhez tartozó teszt eseteket.
- A fejlesztők megkezdik a fejlesztést.
- A teszt menedzser létrehozza a tesztelési tervet, amiben összegyűjti az adott teszteléshez szükséges teszt eseteket.
- A fejlesztés végeztével a tesztelő leteszteli a lefejlesztett funkciókat.
- Amennyiben szükséges, a tesztelő elkészíti a hiba jelentéseket.
- A fejlesztők kijavítják a hibákat, és újbóli tesztelésre bocsájtják azokat.
- Amennyiben nincsen további hiba, úgy a tesztelő megtartja a tesztelést közösen az üzleti ügyfelekkel.
- Az üzleti ügyfelek által, ha a tesztelési eredmény elfogadható, élesítésre kerülnek az új, és javított funkciók.



4.1 ábra: A tesztelői és fejlesztői munkafolyamat módosított ábrája

4.2 Tesztrendszer fejlesztési terve

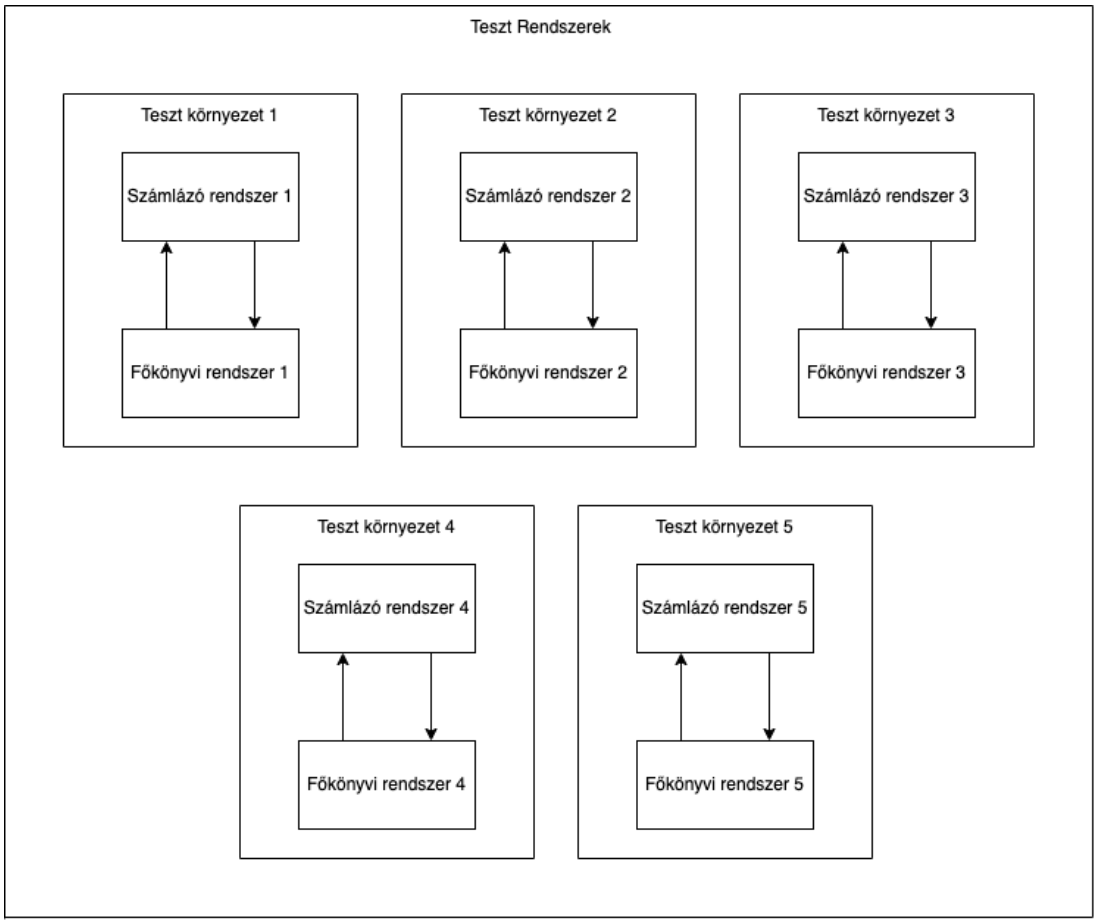
Az anomáliák kiküszöböléséhez leginkább alkalmas megoldás, hogy a tesztelői környezeteket el kell szeparálni egymástól. Ezt az általam elkészített [*4*.2 táblázat] mutatja be:

Teszt Környezet	**Rendszerek**	**Felhasználók**
Fejlesztői környezet	Számlázó rendszer 1 – Főkönyvi rendszer 1	Fejlesztők
Tesztelői környezet	Számlázó rendszer 2 – Főkönyvi rendszer 2	Tesztelők
Üzleti környezet 1	Számlázó rendszer 3 – Főkönyvi rendszer 3	Üzleti ügyfelek
Üzleti környezet 2	Számlázó rendszer 4 – Főkönyvi rendszer 4	Üzleti ügyfelek
Automatizált Teszt környezet	Számlázó rendszer 5 – Számlázó rendszer 5	Tesztelők

4.2 táblázat: Módosított teszt környezetek szeparációja

A rendszerek oszlop jelöli a rendszerek szeparációját. Minden teszt környezeten van egy Számlázó rendszer, és egy ahhoz tartozó Főkönyvi rendszer. A rendszerek közötti átjárás nem lehetséges, így a rendszerek közötti alkalmazások keresztbe kötése sem megvalósítható. Ennek oka, hogy el tudjuk kerülni az anomáliákat.

A biztonsági mentéseket is függetlenítettem a teszt környezettől. Ennek oka, hogy az adatvesztéssel járó kockázatokat igyekeztem minimalizálni. Ha a biztonsági mentések továbbra is részét képeznék a teszt környezetnek, illetve a tesztrendszernek, az súlyos anyagi veszteséget is okozhatna a vállalatnak, illetve olyan adatok elvesztésének a lehetőségét hordozná magában, amik nem, vagy csak nagyon nehezen lehetnének pótolhatóak. Ezt szemlélteti a [*4*.3 ábra] amit a draw.io alkalmazásban készítettem el [5]:



4.3 ábra: A tesztrendszer módosított ábrája

Az átalakítás miatt nehézséget okozhat egy új teszt környezet létrehozása. Amíg a másolás még könnyen megoldható, addig az adott teszt környezeten belüli beállítások időigényesek. Ilyen beállítás például a teszt felhasználók jogosultságainak beállítása. Az erre született megoldás egy script létrehozása, ami a felhasználók jogosultságait automatikusan lemásolja az újonnan létrehozott teszt környezetbe az éles rendszerről, vagy másik teszt környezetről. Így a teszt környezetek létrehozása bizonyos értelemben automatizálva lesz.

4.3 Hiányzó dokumentumok fejlesztési terve

A tesztelői dokumentumok létrehozása a tesztelés egész folyamatát végig követik. Ezekből a dokumentumokból a legfontosabb a Tesztelési terv, Teszt esetek, Hiba jelentés, és a tesztelésről készült összesítő jelentés.

A tesztelőket, illetve a teszt csapatot már a projekt indulásakor be kell vonni a fejlesztési feladatokba, és folyamatokba, hiszen minél korábban történik a bevonásuk, annál kisebb költséggel ki lehet javítani a felmerülő hibákat. [1] Az elsődleges hiba felfedezés már a követelmény meghatározásakor kiszűrhető, ezért a követelmény megfogalmazásánál célszerű, ha jelen van a tesztelő is, hiszen a tesztelő fogja megírni a teszt eseteket, és a további tesztelői dokumentációt, ami szorosan összekapcsolódik a fejlesztés validációjával. Erre hivatkozik az ASPICE is a V-modellt alapul véve. [2]

Az aktuális fejlesztések ezeken a rendszereken azonban agilis módszertannal történnek, de vegyíteni lehet az ASPICE metodológiával, ami kimondja, hogy a projekt, és fejlesztés indulásával párhuzamosan meg kell kezdeni a tesztelői munkák előkészületeit, és el kell készíteni a dokumentációkat.

Első ilyen dokumentáció a teszt esetek létrehozása követelmény ellenében, amik a követelmény létrehozását követően egyből megvalósíthatóak. A mérföldkövek meghatározása után a tesztelési tervet, és a teszt stratégiát is létre kell hozni, ahol definiálom a tesztelési adatokat, módokat és ütemtervet.

A hiányzó dokumentumok fejezetben említett dokumentumok elkészítése az itt említett metódus szerint lesz végrehajtva.

5. Dokumentum formázó alkalmazás ismertetése

Nagyvállalatoknál, ahol informatikai, és egyéb fejlesztések is folynak, fontos részét képezik a projektek, és további fontos információk dokumentálása. Erre a célra a vállalatok különböző alkalmazásokat használnak. Az adott vállalat, melynél én is dolgozom, a dokumentációkat a Microsoft Word nevezetű alkalmazásban készítik el.

Ez az alkalmazás lehetőséget biztosít, hogy bizonyos, iparhoz köthető szabványoknak megfelelő, írott dokumentumokat lehessen létrehozni. A vállalati auditok során az ellenőrző szervezet szigorú szabályokat állít fel arról, hogy egy adott dokumentum milyen struktúra, felépítés, tartalmi, és formai megkötés alapján jöhet létre, és létezhet.

Általános probléma, hogy azonos típusú dokumentumok, mint például követelmények dokumentációja, nem azonos formázással készülnek el, és vállalaton belül a különböző projektek követelmény dokumentumaiban eltérés tapasztalható. Ez az auditáló szervezet számára nem elfogadható. Számos ilyen ipar létezik a piacon, mint például: autóipar, légitársaságok, gyógyszer cégek.

Az általam használt alkalmazás a cég saját fejlesztése, és a Microsoft Wordbe integrálható. Így a Microsoft Word kelléktára kiegészül számos további funkcióval. A funkciók olyan egységes formázási lehetőségeket biztosítanak a felhasználó számára, aminek a segítségével megoldható az egységesített, és precízen elkészített dokumentációk elkészítése. A funkciók UI felületen használhatóak.

Az alkalmazás tesztelésének automatizálása C# nyelven megvalósítható, ahol lehetőség van az alkalmazást platform függetlenül tesztelni, illetve az egyéb funkciókat, és a beállítások ellenőrzését automatizált tesztekkel végrehajtani. „Az automatizált tesztek emberi beavatkozás nélkül hajtódnak végre. Technikai hátterük változatos: szkriptek, programozási nyelvek, speciális segédprogramok. Az automata tesztek gyorsan, nagy mennyiségű tesztet és tesztadatot képesek kezelni, így olyan feladatokra is alkalmasak, melyekre a manuális tesztek nem. Hátrányuk, hogy speciális technikai ismereteket igényelnek és a karbantartásuk is időigényesebb.” [10]

Az alkalmazás továbbfejlesztése is folyamatban van. Ennek kiegészítő neve „Generic”, azonban további részleteket az alkalmazás nevéről nem közölhetek jelen dolgozatban, mivel céges titoknak minősül, és a termék még nem elérhető a piacon. Azonban a tesztelés automatizálása a Generic nevezetű terméken kerül kialakításra.

5.1 Az alkalmazás első verziójának tesztelése

A tesztelés minden esetben követelmények ellenében történik. Különösen fontos, hogy minden teszt esetnek legyen előzetesen ledokumentált funkcionális, vagy nem funkcionális követelménye. A követelmények tárolása, és dokumentálása egy kifejezetten erre a célra kijelölt alkalmazásban történik, és ebben az alkalmazásban történik minden teszt eset hozzácsatolása a már megírt követelményekhez.

Két fő tipusa van:

1. Manuális tesztelés,
2. Automatizált tesztek futtatása.

A manuális tesztelés kézzel történik egy tesztelő által. Ezen belül az elvégzett tesztelés egyik formája a funkcionális tesztelés, ahol a funkciók ellenőrzését teszteli a tesztelő. „A funkcionális tesztek a szoftver előre meghatározott/dokumentált, konkrét funkcióit vizsgálják. Ezek a tesztek tipikusan ún. feketedoboz tesztek, amelyek a szoftver belső struktúrájának ismerete nélkül, mintegy „kívülről szemlélődve" kerülnek végrehajtásra. A funkcionális teszt azt vizsgálja, amit a szoftver csinál. A funkcionális tesztek alapja mindig egy dokumentum. Ez legrosszabb esetben egy felhasználói kézikönyv, ám erre a célra külön dokumentumokat, ún. funkcionális specifikációkat célszerű készíteni. Ezek a leírások ugyanis olyan technikai információkat, illetve folyamatleírásokat is tartalmaznak, melyekre egy átlagos végfelhasználónak ugyan nincsen szüksége, ám a tesztelés szempontjából életbe vágó lehet. Egy ügyfélnyilvántartó szoftver esetében külön leírás szólhat például egy új ügyfél felvételéhez szükséges lépésekről, de akár arról is, hogy a folyamat során az adatok mely rendszereken mennek keresztül, hol és hogyan tárolódnak stb. Egy kellő részletességű leírás alapján elkészíthetőek a teszt esetek, tesztlépések. Funkcionális teszteket bármely tesztszinten végre lehet hajtani." [8]

A másik a nem funkcionális tesztelés, ahol a tesztelő az alkalmazásban végrehajtott utasítások időbeli válasz reakcióját méri, és a felhasználói élményt. A nem funkcionális tesztelés „egy komponens vagy rendszer funkcionalitáshoz nem kapcsolódó tulajdonságainak tesztelése, mint például megbízhatóság, hatékonyság, használhatóság, karbantarthatóság és hordozhatóság". [7] „A nem-funkcionális teszt a szűkebb-tágabb értelemben vett szoftver számszerűsíthető ismérveit vizsgálja; azt elemzi, hogy az adott szoftver valamit hogyan csinál, vagy valamilyen feltételnek milyen mértékben felel meg. Tipikus nem-funkcionális tesztek: teljesítményteszt, terheléses teszt, használhatósági teszt, megbízhatósági-, hordozhatósági teszt stb. Nem-funkcionális teszteket bármely tesztszinten végre lehet hajtani.A nem-funkcionális teszt a szűkebb-tágabb értelemben vett szoftver számszerűsíthető ismérveit vizsgálja; azt elemzi, hogy az adott szoftver valamit hogyan csinál, vagy valamilyen feltételnek milyen mértékben felel meg. Tipikus nem-funkcionális tesztek: teljesítményteszt, terheléses teszt, használhatósági teszt, megbízhatósági-, hordozhatósági teszt stb. Nem-funkcionális teszteket bármely tesztszinten végre lehet hajtani." [9]

A manuálisan elvégzett funkcionális tesztek esetében minden alkalommal black box [6] tesztelés történik. A black box tesztelés olyan „eljárás, melynek során funkcionális vagy nem-funkcionális specifikáció, komponens vagy rendszer elemzése alapján, a program belső szerkezetére történő hivatkozás nélkül történik a tesztesetek származtatása és/vagy kiválasztása". [6]

A manuális tesztelés többféle tesztelési szinten történik. Tesztelési szintek a következőek:

1. Integrációs teszt
2. Rendszerteszt
3. Acceptance teszt, más néven elfogadási teszt

„Az integrációs tesztek a komponensek egymás közötti, vagy az operációs rendszerrel, csatolófelületekkel stb. történő sikeres együttműködésének a vizsgálatát célozzák. Különösen nagyobb rendszereknél fontos a fokozatos, egymásra épülő tesztelés, mert átgondolatlan vagy hanyag megközelítéssel nehéz és tovább tarthat a hibák valódi okának megállapítása, ami jelentős és felesleges költségeket okozhat.Az integrációs tesztek a komponensek egymás közötti, vagy az operációs rendszerrel, csatolófelületekkel stb. történő sikeres együttműködésének a vizsgálatát célozzák. Különösen nagyobb rendszereknél fontos a fokozatos, egymásra épülő tesztelés, mert átgondolatlan vagy hanyag megközelítéssel nehéz és tovább tarthat a hibák valódi okának megállapítása, ami jelentős és felesleges költségeket okozhat.” [11]

Ahogyan az a definíció szerint olvasható, a tesztelés ezen szintje azért fontos, hogy lássuk, a komponensek közötti kapcsolat megfelelően működik-e. Ez az alkalmazásban a Microsoft Word, és az alkalmazás közötti komponensek kapcsolatát ellenőrzi.

A következő tesztelési szint a rendszerteszt. „A rendszertesztek az éles működésű rendszert hivatottak modellezni és vizsgálni az ahhoz minél jobban hasonlító tesztkörnyezetben. Éles rendszeren végzett átfogó rendszerintegrációs tesztelés (például egy új rendszer bevezetése) szóba sem jöhet! A rendszertesztek a rendszer egészének a működését vizsgálják, kevésbé az egyes funkcionalitásokra.” [12]

Az alkalmazás esetében a rendszerteszt nem egy dedikált tesztkörnyezetben történik, hanem a tesztelő számítógépén, és azon belül a saját Microsoft Word alkalmazásába telepített környezeten. A tesztelő maga telepíti a tesztelni kívánt verziót, és az előre összeállított teszt eseteken megy végig, amit a tesztelési terv tartalmaz.

„A rendszertesztek után többféle, ún. elfogadási teszt végrehajtására is sor kerül. Ezen tesztek célja annak megállapítása, hogy a szoftver (rendszer) valóban képes ellátni a feladatát az elvárt módon. Ezeket hivatásos tesztelők, külső hivatásos tesztelők, vagy maguk a megrendelők, esetleg végfelhasználók végzik. Az elfogadási tesztek típusai:

- Felhasználói elfogadási teszt: ún. üzleti felhasználók végzik.
- Üzemeltetési elfogadási teszt: a szoftver üzemeltetése során tipikusan felmerülő folyamatok tesztelése (pl. visszaállíthatóság)
- Szerződésalapú elfogadási teszt: ezen teszt alapja a fejlesztési projektet megelőzően kötött szerződés, melyben bizonyos szabályozásokat, elvárásokat fogalmaztak meg. Ilyenkor ezek ellenőrzését végzik el.

- Alfa teszt: a potenciális végfelhasználók (teljesen hétköznapi felhasználók, nem hivatásos tesztelők) fejlesztőközpontban végrehajtott tesztjei.
- Béta teszt: szintén végfelhasználók által végrehajtott tesztek, melyet általában mindenki a saját otthonában, ill. saját eszközein végezhet.” [13]

Az elfogadási tesztet az ügyfél, vagy más néven megrendelő végzi, ahol a cégünk alkalmazottai nincsenek jelen. A tesztelés eredményéről írott dokumentumban kapunk visszajelzést, illetve a jegykezelő rendszerben az ügyfél által rögzített hibák nyújtanak tájékoztatást a megrendelő által végrehajtott tesztelés eredményéről.

5.2 Az alkalmazás első verziójának tesztelése, annak hiányosságai és problémái

Az alkalmazás első verziójának tesztelése manuálisan, és unit tesztekkel történik. A unit tesztek sok esetben hiányosak, és az alkalmazás bizonyos részeire nincsenek megírva. A unit tesztek megírása a fejlesztőkre háruló feladat, de előfordul, hogy azt a tesztelő készíti el. A tesztelőre hárul a feladat, hogy manuálisan tesztelje le a funkciókat úgy, hogy előzetesen unit tesztekkel az nem került leteszteléshez.

Hátránya ennek a jelenlegi tesztelésnek az, hogy a tesztelési idő szűkössége miatt nem mindig marad elég idő a kimerítő tesztelésre, vagy a regresszió tesztelésre, és limitálni kell a tesztelendő teszt esetek számát.

Ebből adódik, hogy a szoftver hibamentessége csökken, és gyakori, hogy az ügyfelek találják meg azokat a hibákat, amit egy alaposabb belső tesztelés alkalmával a tesztelők kiszűrhetnének.

További probléma, hogy a hiányzó tesztelési dokumentációk miatt, illetve az idő rövidsége miatt a Support csapat is részt vesz a tesztelésben, és előfordul, hogy a funkciók tesztelése helytelenül van végrehajtva, és a hibák dokumentálása nem írásban történik, hanem szóban. Tehát nincsen alapos hiba jelentés a fejlesztők számára, ezért a nem dokumentált hibák nem minden esetben kerülnek kijavításra.

Az alkalmazás első verzójának automatizált tesztelésének fejlesztése erőforrás hiány miatt egyelőre nem lehetséges. A projekten dolgozó tesztelő egy személyben írja meg a manuális teszt eseteket, felveszi a tesztelés közben talált hibákat, elvégzi a kijavított hibák újratesztelését, és foglalkozik az automatizált tesztek fejlesztésével. Utóbbira marad a legkevesebb ideje, mert a release 3 havonta esedékes, és a sok követelmény lefedése teszt esetekkel nagy erőforrást igényel.

Az alkalmazás funkcionális tesztelésének automatizálása a termék komplexitása, és erőforrás hiány miatt jelenleg nem kivitelezhető, az minden esetben manuális beavatkozást igényel.

5.3 Megoldási javaslat a problémák kijavítására

Legfontosabb lépés a hiányzó dokumentumok pótlása, és a teszt automatizálás bevezetése. Ezzel a két lépéssel megoldást nyújtunk az időhiányra, a hibamentesség növelésére, és az erőforrás problémáira.

A hiányzó dokumentumok pótlásának első lépése a traceability matrix létrehozása, ahol a teszt esetekkel le nem fedett követelményekről kapunk információt. Így láthatjuk, hogy mely követelmények lefedése szükséges még teszt esetekkel. Ezt követően rangsoroljuk a követelményeket. A rangsorolás a funkciók gyakori felhasználása szerint történik, a leggyakrabban használt funkciótól haladunk a legkevésbé használt felé. Figyelembe vesszük, hogy a rendszer működését melyik funkció milyen mértékben befolyásolja, és milyen mértékben van függőségi viszonyuk. Azoknál az eseteknél ahol például az alkalmazás indulását, vagy működését befolyásolja egy funkció, a fontossági listán előre kerül.

A megírt teszt eseteket a tesztelések alkalmával fel tudják használni olyan személyek is, akik nem tesztelőként dolgoznak, de a tesztelésben részt vesznek. Ilyen személy lehet a megrendelő is, akinek szüksége van sorvezetőre ahhoz, hogy saját környezetében megfelelően le tudja tesztelni az alkalmazást, és meggyőződhessen a funkciók helyes működéséről. A megrendelő nem végez nem funkcionális tesztelést.

A hibajelentés létrehozása is könnyebbé válik, hiszen egy teszt eseten belül, adott lépésnél, ahol a hiba jelentkezett, lehetőség van jelölni a teszt eset sikertelen futtatásának eredményét.

A teszt automatizálás bevezetésével a már előre összekészített teszt esetek listájának futtatására van lehetőség, párhuzamosan a manuális tesztek futtatásával. Így ugyan annyi idő alatt jelentősen több funkciót tudunk ellenőrizni/tesztelni, ezért növelni tudjuk az alkalmazás hiba mentességét.

Előnye, hogy az automatikus futtatást követően egyből elérhetővé válik a futtatási eredmény, és nem manuálisan kell előállítani. Ezzel is időt, és erőforrást takarítunk meg.

5.4 Automatizált tesztelés megvalósításának lehetőségei

Az automatizált tesztelést alkalmazások segítségével lehet végrehajtani, de mielőtt számos alkalmazást megvizsgáltam volna, hogy melyik lenne a leginkább alkalmas az alkalmazás tesztelésére, a cég vezetősége úgy döntött, hogy C# alapú tesztelést kell végrehajtanunk, ami saját Framework-el rendelkezik.

Azért született ez a döntés, mert az alkalmazás is C# nyelven íródott, és nem csak UI (más néven frontend) teszteléseket szeretnénk végrehajtani, hanem háttér tesztelést is (más néven backend tesztelés). A döntést befolyásoló további tényezőkről nincs tudomásom. Vezetőségi döntés függvényében a automatizált tesztek bevezetése az alkalmazás továbbfejlesztett verzióján hajtható végre, mely a jelenlegi rendszerre épül.

6. Dokumentum formázó alkalmazás továbbfejlesztett verziójának bemutatása

Az előző verziója az alkalmazásnak platform függő. Emiatt csak olyan ügyfelek számára lehetséges az alkalmazás használata, akik Windows operációs rendszerrel rendelkeznek.

Az alkalmazás tovább fejlesztett verziója egy konzolos applikáció, ami platform független, így használható MacOS, és Linux operációs rendszerekkel is. Jelenleg is fejlesztés alatt áll.

A dokumentum formázó alkalmazás első verziójának komponenseire épül a továbbfejlesztett változat, és szintén dokumentum formázásra alkalmas. Kétfajta komponenst használ fel az előző applikációból. Egyik az offline business logic, aminek a neve OfflineBusinessLogic.dll, a másik pedig az offline modulok.

A BusinessLogic.dll-t azért használja az applikáció, mert az absztarkciók, és szabályok itt vannak definiálva. Továbbá innen érkezik offline helper, offline module helper, és offline word helper.

Az alkalmazás offline módban használható, ami azt jelenti, hogy nincs szüksége a Microsoft Word futtatására. Két fő funkciója van. Egyik az inspection, más néven check, másik a correction, más néven fix.

Az inspection funkció feladata, hogy a dokumentumban lévő hibákat detektálja. Ilyen hibák például a dupla space-ek, vagy a fejezet címek nem megfelelő számozása.

A correction funkció feladata, hogy a detektált hibákat automatikusan kijavítsa. Nem minden inspection modulhoz tartozik correction.

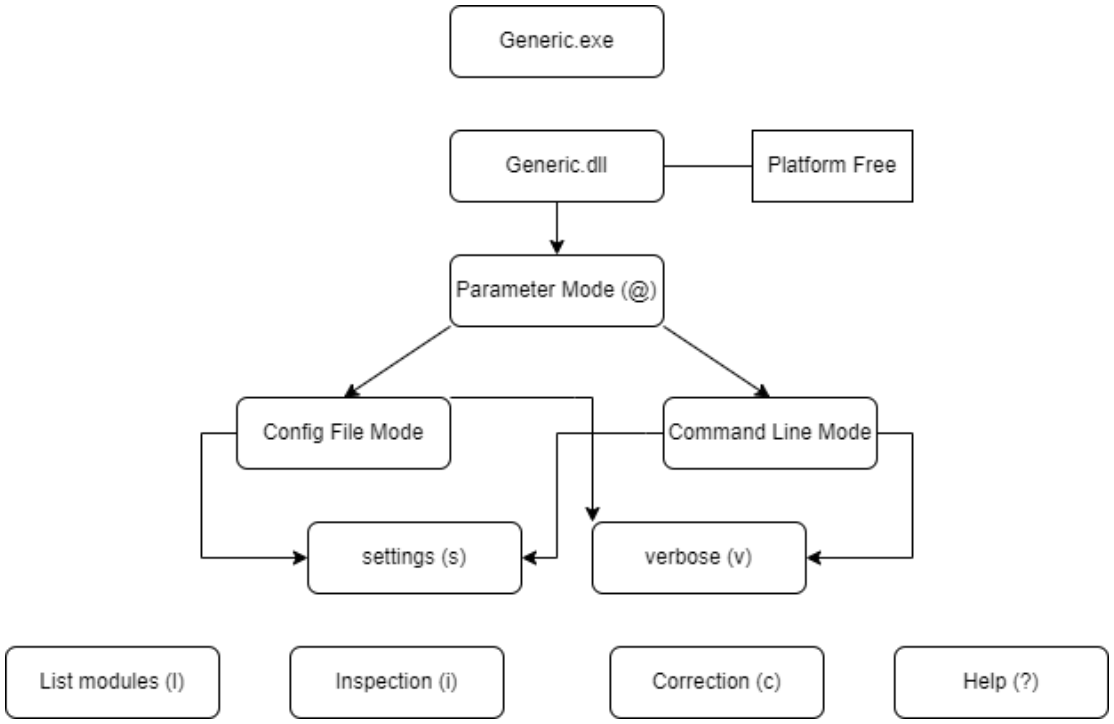
Az alkalmazás a BusinessLogic.dll-t dll-ként használja. A Business kód a korábbi termékben, az első verzióban található, és a dll a továbbfejlesztett alkalmazásba való bemásolás után hivatkozik rá, így úgy használja mint egy 3rd party dll-t.

A modulok használata command line utasításokkal lehetséges, amit a fejlesztők definiáltak a követelmények függvényében.

6.1 A továbbfejlesztett dokumentum formázó alkalmazás architektúrája

Az alkalmazás .exe formátuma Microsoft Windows operációs rendszeren futtatható. Ennek részét képezi a dll, azonban a dll segítségével van lehetőség más platformokon is használni az alkalmazást .exe nélkül. Az architektúrát amit vállalaton belül a vezető fejlesztő készített, majd én megrajzoltam a draw.io [5] alkalmazás segítségével, a [

6.1. ábra] mutatja be:



6.1. ábra: A dokumentum formázó alkalmazás architektúrája

A Parameter Mode-nak köszönhetően tudja az alkalmazás, hogy honnan vegye a futáshoz szükséges információkat, a config file-ból, vagy a command-ból. Továbbá kerül meghatározásra a dokumentum, amin az alkalmazást a felhasználó futtatja, illetve itt kerül definiálásra a kimeneti dokumentum is, és annak neve, amiben a correction által elvégzett javítások találhatóak.

Két lehetőség van az alkalmazás használatára. Az egyik lehetőség command line-ból történik, a másik config file-ból. Minden esetben meg kell határoznia a felhasználónak, hogy milyen módon szeretné használni az alkalmazást. Command line móddal, vagy config file móddal. Az itt definiált utasításokat command-nak hívják. A command határozza meg, hogy az alkalmazás mit csináljon. Ide tartoznak még a feature választók is, amiknek a segítségével meg tudja határozni a felhasználó, hogy melyik feature-ét szeretné választani, és használni az alkalmazásnak, majd ezután következnek az option-ök, amik a command paraméterezését jelentik.

Command line használata egy meghatározott formátum szerint kell történjen. Felépítése a következő: ProgramNeve.exe -@ genericConfig.json – TestFile.docx

1. Az .exe a windows alatt futtatható exe file-t jelenti
2. A -@ paraméter utasítás ami utasítja a programot hogy a .json file-ban definiált utasítások alapján fusson le a TestFile.docx dokumentumon.
3. A genericConfig.json a futtatandó utasításokat tartalmazó file
4. A TestFile.docx az a file, amin az utasítások lefutása történik.

A pédában látható Config file esetében a parancsok .json, vagy .xml formátumban kerülnek definiálásra.

Az utasítások végrehajtása történhet paraméterezett utasítások nélkül is, ahol a .json file-t nem kell megadni, mert a command line használata közben kell beírni az utasítást.

Ezt követően két féle global option meghatározására van lehetőség. A global option egy globális beállítás, ami a teljes futás alatt érvényben marad. Egyik global option a settings, másik a verbose.

A settings-nél Docdescriptive file-okat lehet megadni. A Docdescriptive egy a terméken belüli adathordozó formátum, ami megoldás arra, hogyan csomagoljunk be olyan információkat amiket a termékben lévő egyes tool-ok kivesznek, és használnak.

A verbose egy globális kapcsoló, ami megmutatja, hogy a command line parse hogyan értelmezi a command line-t, az helyes-e, és ha nem, akkor nem futtatja le. Ha a verbose parancsot nem használja a felhasználó, akkor a command line nem ír ki semmit.

A programon belül négy különböző parancsunk (command) van. A modulokat, és a hozzájuk tartozó parancssorokat az általam elkészített [6.2 táblázat] szemlélteti:

List modules	**Inspection modules**	**Correction modules**	**Help modules**
Ectension (-x)	Extension (-x)	Extension (-x)	-l
	Modules (-m)	Modules (-m)	-i
	Source (--)	Source (--)	-c
	Result (-r) arg	Output (-o) arg	?
	Format (-f) arg	Result (-r) arg	
	Filter (-n) args	Format (-f) arg	
	Time (-e)	Filter (-n) args	
	Output (-o) arg	Time (-e)	
	Bookmark (-b)	Bookmark (-b)	
	Highlight (-h) arg	Highlight (-h) arg	

6.2 táblázat: Modulok és a hozzájuk tartozó parancssorok

A list modules parancsa a (-l) kötőjel l. Kapcsolódik hozzá az -x ami az extension-t jelenti. Ezek után van lehetőség a modulok felsorolására. A modulok úgy működnek, mintha erőforrás file-ok lennének, nincsenek belefűzve az alap programba. A lista modul kilistázza az utasításnak megfelelő elemeket a modulokból. Lehetőség van megtekinteni az inspection modulokat, a correction modulokat, és a kettő párosítását is ezzel a paranccsal.

Az inspection, és correction modulok utasításai a modulok tulajdonságainak kiírására szolgálnak.

A help modult az inspection és correction modulokról adott információkat listázza ki.

7. Az automatizált tesztelés bevezetése a dokumentum formázó alkalmazás továbbfejlesztett verzióján

Az automatizált tesztelés bevezetése vezetőségi döntés volt. Az alap termék továbbfejlesztett verziójának tesztelésére nem állt rendelkezésre elegendő erőforrás, ezért költséghatékonyság szempontjából a teszt automatizálás megvalósítása lett a cél. A bevezetés elsődleges szempontja az volt, hogy a tesztelést képesek legyenek olyan személyek is elvégezni, akik nem rendelkeznek tesztelői tapasztalattal.

Azonban az automatizált tesztek fejlesztéséhez, és létrehozásához elengedhetetlen a tesztelői tudás. A követelményeket én és kollégáim értelmeztük, melynek alapján meg tudtuk kezdeni az automatizált tesztek fejlesztését.

Az automatizált tesztek megírása Microsoft Visual Studio-ban történik C# nyelven. A teszt projekt MSTest Test Project (.NET Core)-al [16] lett megvalósítva. Ez egy olyan eszköz, ami a forráskód tesztelését segíti elő annak validálásával és verifikációjával.

7.1 Az automatizált tesztek megvalósítása és felépítése

Az alkalmazás Visual Studio-ban [14] lett megvalósítva C# nyelven. [15] Az alkalmazás Solution-jében van a projekt, ahol a fejlesztők által megírt kódok találhatóak. Emellett a projekt mellett van a Teszt projekt, ami egy MSTest Test Project (.NET Core) [16], és ez lett hozzá csatolva a Solution-höz. Ezen keresztül lehet referenciával elérni a product-ot.

Az MSTest Test Project kiválasztása vezetőségi döntés volt, ezért más lehetőség nem jöhetett szóba.

A tesztek meghatározása teszt osztályokban történik, és minden osztály publikus, hogy elérje a tesztelés során a teszteket. A teszt osztály struktúrája a [7.1 ábra]-n látható:

```
[TestClass]
public class GlobalOptions : TestAncestor
{
    /// <summary>
    /// Run before all test method onnce in this class. All files copied from test documents folder to the execution folder
    /// </summary>
    /// <param name="context"></param>
    [ClassInitialize]
    public static void ClassInitialize(TestContext context)
    {
        CopyFiles(TestDirPath, Environment.CurrentDirectory);
    }

    /// <summary>
    /// Copy files from to. Depend on the ClassInitialize method
    /// </summary>
    /// <param name="sourceDir"></param>
    /// <param name="targetDir"></param>
    private static void CopyFiles(string sourceDir, string targetDir)
    {
        foreach (var file in Directory.GetFiles(sourceDir))
            File.Copy(file, Path.Combine(targetDir, Path.GetFileName(file)), true);
    }

    /// <summary>
    /// ONSG-133
    /// Checks if the main execution part was skipped when executing SyntaxCheck (-v) command
    /// If ExecuteWorkflow of checkerservice is called >> throw exception
    /// </summary>
    [TestMethod]
    public void SyntaxCheck_SkipMainExecutionTest()
    {
        InitVerbosityBoard();
        string[] ArgArray = new string[] { "-v", "-i", "-x", "OfflineModulesGeneral.dll", "-m", "1205", "--", "TestFile.docx" }; // example argArray
        if (genO.Work(ArgArray) == -1) // if any exception caught or ListModules/ExecuteWorkflow got called
        {
            Assert.Fail();
        }
    }

    /// <summary>
    /// ONSG-134
    /// Checks expected output contents and sequence for SyntaxCheck command (-v)
    /// </summary>
    [TestMethod]
    public void SyntaxCheck_OutputTest()
    {
```

7.1 ábra: A teszt osztály felépítése

A teszt osztályon belül egy osztály inicializálást hoztam létre, ami teszteléshez szükséges file-okat átmásolja az automatizált teszt által használt mappába, majd ezt követően a teszt esetek definiálása látható. Minden teszt eset külön teszt metódusban van definiálva.

A osztályok a TestAncestor osztályból vannak származtatva. Erre azért van szükség, mert minden követelmény csoporthoz, a többi teszt osztálytól független teszt osztály kerül létrehozásra. A teszt futtatásánál, és kiértékelésénél van ennek a logikának szerepe, hogy egymástól elkülönítve lehessen dokumentálni a tesztek futási eredményeit, és könnyebb legyen a hiba detekciója. Azonban vannak olyan elemei az egymástól elkülönített teszt osztályoknak, ahol azonos elemeket használnak fel. Ilyen például a [7.1 ábra]-n látható genO.Work(ArgArray), vagy akár a teszt dokumentumok elérése is. Annak érdekében, hogy ezt ne kelljen minden osztályban külön definiálni, az ős osztályban valósul meg a definiálása.

Ahhoz, hogy az automatizált teszteket futtatni lehessen, nem szükséges megnyitni a Microsoft Word alkalmazást. Azonban ha a dokumentumon szükséges javításokat el szeretném végezni, tudnom kell a hiba pontos helyét, és nem csak a hibák számosságát.

A hiba pontos helyét úgy tudom megállapítani, hogy kiszámolom range kalkulációval a hiba elhelyezkedését. A range kalkuláció mindig az inspection hibakeresés, és a javítás között van. Először megtörténik a hibakeresés, majd ezt követően

kalkuláljuk a hibáknak a range-eit, aminek segítségével megtörténik a javítás. A javítás lehet például egy szövegrész kitörlése, amihez tudnom kell, hogy hányadik karaktertől, hányadik karakterig kell kitörölni a hibás szövegrészt.

Az automatizált tesztek futtatása mindig tisztított környezetben kell, hogy történjen. A tisztított környezet, azt jelenti, hogy amikor egy automatizált tesztben generáltam egy dokumentumot, mint kimeneti dokumentum, és ezt a dokumentumot felhasználtam, de már egy következő tesztet futtatok, ami szintén egy kimeneti dokumentumot fog generálni, akkor a meglévő kimeneti teszt dokumentumot törölni kell. Ezt a törlést két módszerrel tehetem meg.

Egyik módszer, hogy az osztályon belül definiálom a törlést úgy, ahogyan a forráskódból származó [7.2 ábra]-n látható:

```
[TestCleanup]
public void TestCleanup()
{
    CleanFiles();
    genO.Dispose();
}

[ClassCleanup]
public static void ClassCleanup()
{
    CleanFiles();
}

private static void CleanFiles()
{
    if (File.Exists(OutFilePath))
    {
        File.Delete(OutFilePath);
    }

    if (File.Exists(ResultPath))
    {
        File.Delete(ResultPath);
    }
}
```

7.2 ábra: Az osztály és metódus Clean up-ja

Tehát a TestCleanup minden alkalommal le fog futni egyszer minden olyan teszt metódus előtt, amit az adott teszt osztály tartalmaz, és a ClassCleanup minden alkalommal le fog futni egyszer, mikor a teszt osztály futtatása megtörténik. Így törlődnek

azok a fájlok, melyeknek használatára a következő teszt metódus futtatásánál már nincsen szükség.

Másik módszer amit a forrsákódból fotóztam ki a [7.3 ábra]-n látható:

```
[ClassCleanup]
public static void ClassCleanup1()
{
    Directory.Delete(Directory.GetCurrentDirectory());
}
```

7.3 ábra: Másik módszer a Class Clean up-ra

Ez a ClassCleanup minden esetben azt a mappát törli ki, ahol a teszt létrehozott fáljokat. Ebben az esetben célszerű egy külön mappát meghatározni, ahova a teszt létrehozza a tesztelési fájlokat.

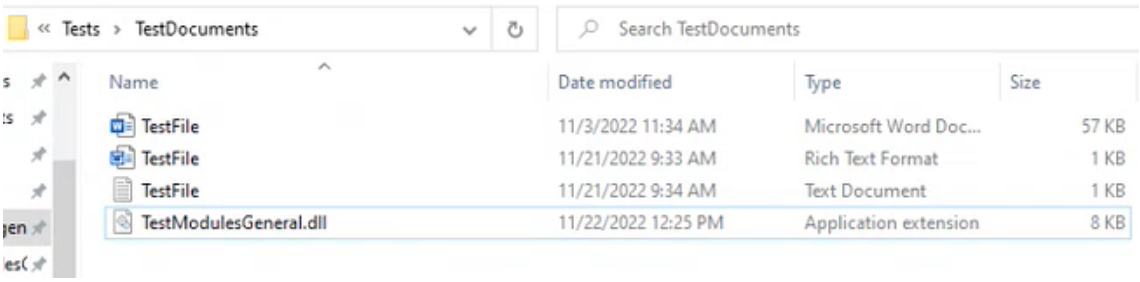
Ugyan ilyen Cleanup létre hozható a TestClenaup-al is, ami minden teszt metódus futtatása előtt elvégzi a törlést. Ebben az esetben nem egy teljes teszt mappa kerül törlésre, hanem a teszt metódus futtatása alkalmával létre hozott fájlokat törli ki a Cleanup azért, hogy a következő teszt metódus futtatását ne befolyásolja az előzőleg lefutattott teszt metódus futtatás eredménye.

A teszt futtatását megelőzően szükség van arra, hogy a teszteléshez szükséges dokumentumok rendelkezésre álljanak abban a mappában, ahol a teszt dolgozik. Emiatt egy olyan metódust hoztam létre, ahol a saját mappámból a program bemásolja a megfelelő file-okat oda, ahonnan a teszt eléri azokat. Ez a [7.4 ábra]-n látható:

```
/// <summary>
/// Copy files from to. Depend on the ClassInitialize method
/// </summary>
/// <param name="sourceDir"></param>
/// <param name="targetDir"></param>
private static void CopyFiles(string sourceDir, string targetDir)
{
    foreach (var file in Directory.GetFiles(sourceDir))
        File.Copy(file, Path.Combine(targetDir, Path.GetFileName(file)), true);
}
```

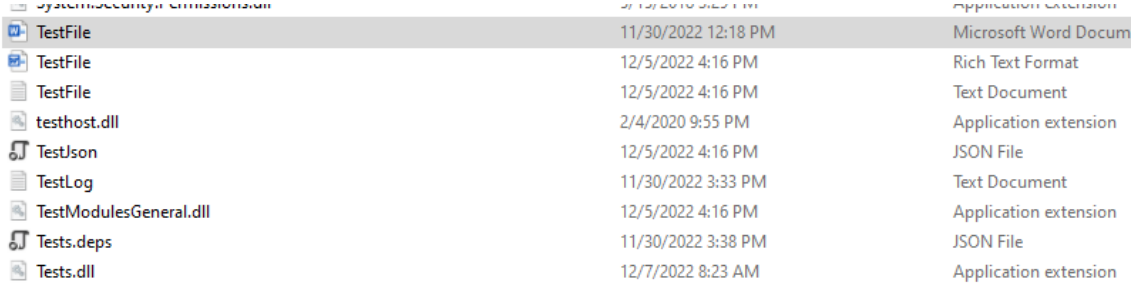
7.4 ábra: Metódus a file-ok másolására

A forrás mappa tartalma a [7.5 ábra]-n látható:



7.5 ábra: Teszt file-ok forrás mappája

Az összes ábrán látható file átkerül a teszt mappájába, ami a [7.6 ábra]-n látható:



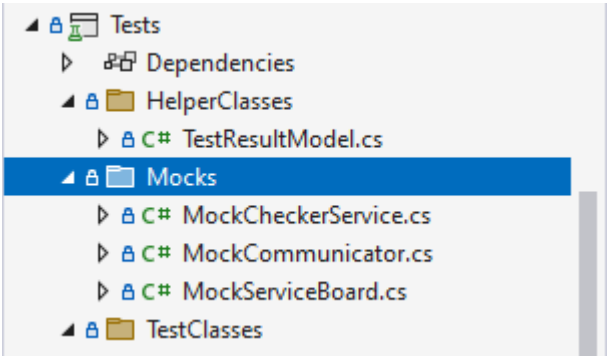
7.6 ábra: A teszt dokumentumok cél mappájának tartalma a másolás után

Azért fontos a TestFile dokumentumok megléte, mert az automatizált tesztek futtatásának alkalmával ezeken a dokumentumokon fog dolgozni az automatizált teszt. Ezeket a dokumentumokat fogja módosítani, és a futtatás alatt létrehozza azt a kimeneti dokumentumot, amit egy következő teszt metódus futtatásánál törölni fog.

7.1.1 Mocks

„A mocking egy olyan folyamat, amely lehetővé teszi egy olyan mock objektum létrehozását, amely egy valódi objektum viselkedésének szimulálására használható. A mock objektum segítségével ellenőrizheti, hogy a valós objektumot a várt paraméterekkel hívta-e meg, és ellenőrizheti, hogy a valós objektumot nem váratlan paraméterekkel hívta-e meg.” [17]

Az automatizált tesztelés keretein belül megvalósított Mock tesztek struktúrája a [7.7 ábra]-n látható:

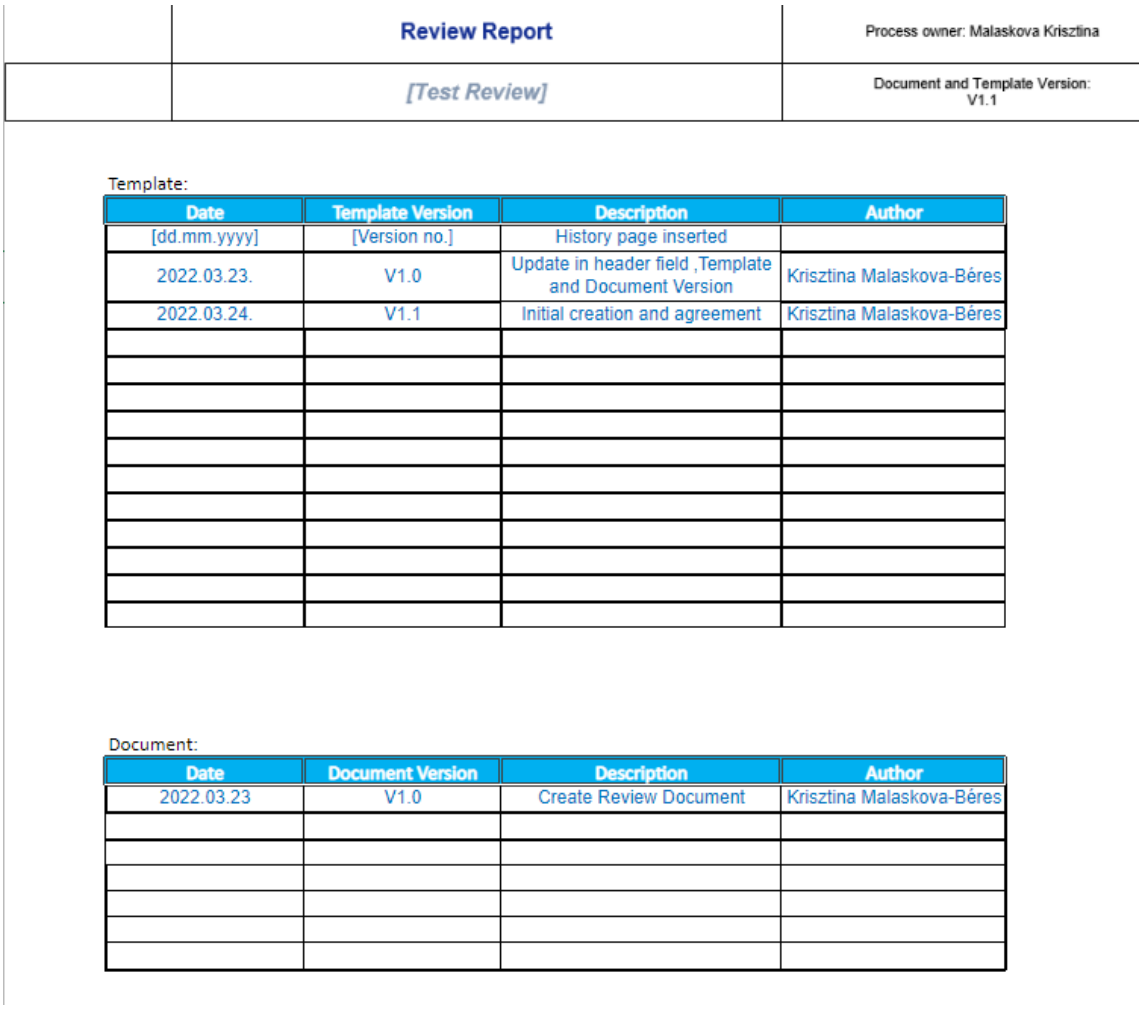


7.7 ábra: Mocks a tesztelés struktúrájában

Ennek tesztelését a tesztelők végzik, azonban a fejlesztők végezték a fejlesztését, ezért a szakdolgozatom további részében nem térek ki rá részletesebben. Azért tartottam szükségesnek megemlíteni, mert az automatizált tesztek futtatásánál minden esetben a tesztelő végzi a mock tesztek futtatását, és kiértékelését.

7.1.2 Az automatizált tesztek review-ja

A review, más megnevezésén ellenőrzés szintén fontos szerepet játszik a hatékony teszt automatizálásban. Jelenleg az automatizált tesztek kód ellenőrzését a fejlesztők végzik, de szükséges a tesztelők bevonása is. Minden kód ellenőrzés alkalmával kötelezővé kell tennem a review dokumentum kitöltését. Ennek sablonját már elkészítettem, ami a [7.8 ábra]-n látható:



	Review Report	Process owner: Malaskova Krisztina
	[Test Review]	Document and Template Version: V1.1

Template:

Date	Template Version	Description	Author
[dd.mm.yyyy]	[Version no.]	History page inserted	
2022.03.23.	V1.0	Update in header field ,Template and Document Version	Krisztina Malaskova-Béres
2022.03.24.	V1.1	Initial creation and agreement	Krisztina Malaskova-Béres

Document:

Date	Document Version	Description	Author
2022.03.23	V1.0	Create Review Document	Krisztina Malaskova-Béres

7.8 ábra: Review dokumentum első oldala

A sablon kezdő képernyője csak a verziók számát, felelősét, és dátumát tartalmazzák. Ami ennél sokkal fontosabb az a review adatainak dokumentálása, és a talált hibák feljegyzése. A rendszer amit eddig felépítettem a tesztelés review-jával kapcsolatban, az a dokumentum struktúra.

A review adatai a [7.9 ábra]-n láthatóak. Az általános információknál ki kell tölteni a projekt nevét, a review megnevezését, mint például az automatizált tesztek review-ja, a review típusát, amit az ISTQB [4] szabvány is definiál, illetve a review dátumát.

Ezt követően meg kell adni a dokumentum adatait, hogy mi az a lista, teszt, vagy követelmény amit review-zni szükséges. A résztvevők megnevezése, és felkszülési idejük is meghatározásra kerül. A review azért fontos, mert egy esetleges audit alkalmával az auditot végző szervezet bekérheti azokat a dokumentumokat, amik a minőségbiztosítást szolgálják. Amennyiben nincsen evidencia arra, hogy a tesztelést, és teszt fejlesztést megelőzte egy biztonsági kapu, úgy az auditáló bizottság joggal állíthatja azt, hogy a

minőségbiztosítás nem megfelelő, hiszen a tesztek futtatása, és az azt megelőző folyamatok ellenőrzésére nincsen bizonyíték.

Ezt kikerülve szükséges a review folyamat összes adatát egy hivatalos dokumentumban tárolni.

General			
Project			
Review name	Review of Automation Test Cases		
Review type	Walkthrough		
Review criteria			
Documents			
Sr. no. P#	Test object	Version	Date
D1		*V1.0*	*2022.03.28.*
D2			
D3			
Sr. no. R#	Reference documents	Version	Date
D4			
D5			
Participants			
Author	Name		Preparation time [h]
A1			
A2			
Reviewer	Name		Preparation and review time [h]
R1	*Malaskova Krisztina*		
R2			
R3			
R4			
Moderator	Name		Preparation time [h]
M1	*Malaskova Krisztina*		
Client	Name		Preparation time [h]
C			
Deadlines			
Deadlines	Participant	Date	Meeting duration [h]
Result			
Summary of findings			
Repeat review on			
Review closure / Release of review object			
Name	Remarks	Date	Comment
Statistics			
	Effort for meetings [h] (Duration of all meetings * Number of participants		
	Effort for individual preparation and review performing [h] (sum)		
	Number of reviewed lines of code		

7.9 ábra: A review-hoz szükséges adatok

A review dokumentumban szükséges az összes teszt eset feljegyzése, azok prioritási szintjei, illetve lokációi. Ez a [7.8 ábra]-n látható:

List of findings					
Sr. no.	Jira ID	Owner	Classification	Comment	Type
1		A1	Low		Manual
2		A1	Medium		Manual
3		A2	High		Automation
4		A2	Critical		Automation
5		A1			
6					

7.8 ábra: Teszt esetek rögzítésének felülete

A lokáció a Management Tool-ban kiosztott ID alapján azonosítható. Minden teszt esetnek van egy saját tulajdonosa, aki a teszt fejlesztését elvégezte, és a teszt esethez a megfelelő fontossági szintet hozzá rendelte, és megjelölte a teszt típusát.

A review folyamán talált hibákat a találatok menüpontban lehet dokumentálni, ami a [7.10 ábra]-en látható:

List of findings							
Sr. no.	Jira ID	Page/line, chap, or the like	Reviewer	Classification	Description	Actions	Status
1			R1	Low	Typo in second line	Accepted	Open
2			R2	Medium	Missing login option	Declined	Closed
3			R3	High		Not Relevant	In Progress
4			R2	Critical			Blocked
5			R5				Postpone
6							
7							

7.10 ábra: Az észrevételek rögzítésének felülete

A táblázatba pontosan be kell írni a hiba detektálásának helyét, a személyt, aki a hibát megtalálta, a hiba leírását, a hiba javítására vonatkozó állapotot, és a hiba találatra vonatkozó értékelést. Ilyen értékelés például az, ha a review-t végző csapat a hibát elfogadja, és valóban hibának nyilvánítja, de el is utasíthatják, ha a döntés szerint az észrevétel nem minősül hibának.

7.2 Az automatizált tesztek megvalósítása közben felmerült problémák

Az automatizált tesztek fejlesztése közben szembesültem azzal a problémával, hogy nem tudom hiba nélkül lefuttatni a kódot, mivel a program a kimeneti teszt file-okat a ProgramFiles dotnet mappába szeretné betenni. Azonban a ProgramFiles dotnet mappa a vállalaton belül írás védett, és még adminisztrátori jogosultsággal sem módosítható, mert az adminisztrátori jogok engedélyezését követően a rendszer automatikusan elveszi az adminisztrátori jogosultságokat.

A problémát hiába próbáltam megoldani úgy, hogy a kódban más elérési útvonalat adtam meg, a program továbbra is a ProgramFiles dotnet mappájába próbálta végrehajtani a teszt kimeneti dokumentumának létrehozását.

További probléma volt, hogy az első teszt modul futtatását követően a további tesztek mind hibás eredménnyel futottak le. A hiba üzenetben a tesztelői dokumentumra hivatkozott. Probléma volt a forrás dokumentumokkal, illetve a kimeneti dokumentumokkal is.

7.3 Az automatizált tesztek megvalósítása közben felmerült problémák megoldása

A ProgramFiles dotnet mappájának felülírását nem tudtam a fejlesztői környezeten belül megoldani. Megkerestem az automatizált kód azon részét, ahol definiálva van, milyen mappából kell a tesztelői dokumentumokat kiolvasnia a programnak, és milyen mappába kell a kimeneti dokumentumot beletennie. Ennek útvonalát módosítottam, de a program továbbra is az előzetesen használt rossz mappák felülírását akarta megvalósítani.

Ezt követően a teljes környezetet letöröltem, és újra telepítettem, de a hiba ennek ellenére továbbra is fenn állt. Segítséget kértem a vezető fejlesztő kollégától, de Ő sem tudta a problémát megoldani. A nehézsége a hiba detekciónak az volt, hogy vállalaton belül mindenkinél megfelelően működött az automatizált teszt futtatása, a teszt dokumentumok a megfelelő mappába lettek átmásolva a program által, és a kimeneti dokumentum is az elvárt mappába került a teszt futtatás végén.

Végül a megoldás az lett, hogy a fejlesztői kódban, csak az én környezetemben át lett írva a mappa elérési útvonala, és a teszt metódusokban az Environment.CurrentDirectory hivatkozással a megfelelő mappába kerültek a dokumentumok.

A teszt modulok hibás futtatásának hibáját az okozta, hogy a program a már korábban létrehozott kimeneti teszt dokumentumot nem tudta elhelyezni a kimeneti mappába arra hivatkozva, hogy a dokumentum elnevezése nem megfelelő. A megoldás végül az lett, hogy minden teszt metódust követően a kimeneti dokumentumot törölni kell, így a teszt metódusok egymás után végre hajthatóak, és a program ugyan azon a néven létre tudja hozni a kimeneti dokumentumot.

A forrás dokumentumok esetében a hiba oka az volt, hogy a program által használt mappában nem voltak elérhetőek a teszteléshez szükséges dokumentumok, ezért megírtam a forrás mappából történő dokumentumok cél mappába történő átmásolását.

7.4 Az automatizált tesztek futtatása és eredményei

Az automatizált tesztek futtatását a tesztelők végzik. Az én feladatom, hogy a tesztelés időben elkezdődjön. Ennek előfeltételei a következők:

1. A követelmények meghatározása
2. A követelmények megvalósítása a fejlesztők által
3. A követelmények lefedése teszt esetekkel
4. A megírt teszt esetek lefejlesztése

5. A lefejlesztett teszt esetek mindegyike egy követelményre, és egy írott teszt esetre hivatkozik
6. A lefejlesztett teszt esetek a követelményeknek megfelelően működnek
7. A megfelelő mennyiségű tesztelő áll rendelkezésre a tesztelés idejére
8. A menedzsment biztosítja a megfelelő időt a tesztelésre
9. A fejlesztők nem csúsznak annyit a fejlesztéssel, hogy a tesztelés a biztosított időbe ne férjen bele
10. A teszt környezet előkészítése és biztosítása

A tesztelési idő kiszámítását mindig a release időpontjához igazítom, ezért legalább másfél héttel a release időpontja előtt a tesztelést el kell kezdeni. Azért választottam a másfél hetes intervallumot, mert így van ideje a tesztelőnek lefutattnia a teszteket, ami egy napot vesz igénybe. Ha a tesztek futtatását követően az automatizált teszt elbukik, a tesztelőnek van egy napja arra, hogy a teszt eredményeit kiértékelje. Itt a tesztelő megállapítja, hogy az elbukott tesztek miért futottak hibára.

A hibás teszt eredményeknek több oka is lehet, amit a következő fejezetben fejtek ki bővebben.

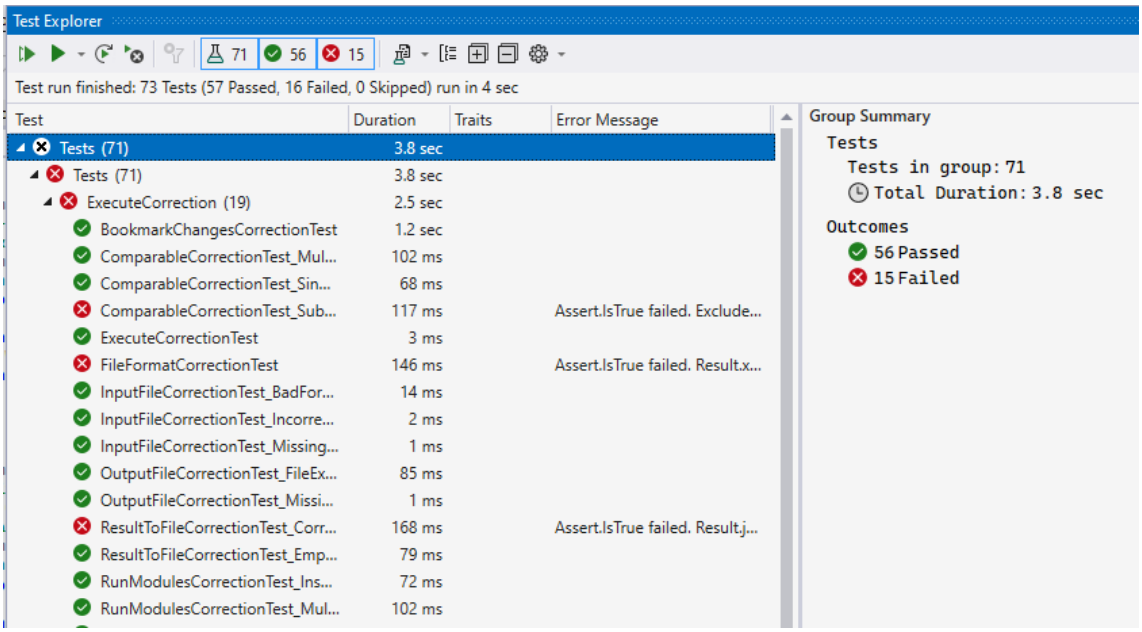
Amennyiben a teszteket javítani kell, úgy erre kettő nap áll rendelkezésre. Itt van lehetőség a hibásan megírt automatizált tesztek javítására, és a technikai hibák elhárítására. Ezt követően az automatizált tesztek újboli futtatása szükséges.

7.4.1 Az automatizált tesztek futtatása a gyakorlatban

Az automatizált tesztek futtatása a Microsoft Visual Studio 2022 Test Explorer funkciójával történt. A futtatás végeztével voltak olyan automatizált tesztek, amik hibás eredményel futottak le. A hibás eredmények okai nem a program hibájából adódtak. Az automatizált tesztek ős osztályának tartalma módosítva lett. A módosítás következménye, hogy a teszt metódusokban definiált elemek hivatkozása már nem megfelelő. Ennek kijavítására létrehoztam egy ütemtervet, és szabályzatot.

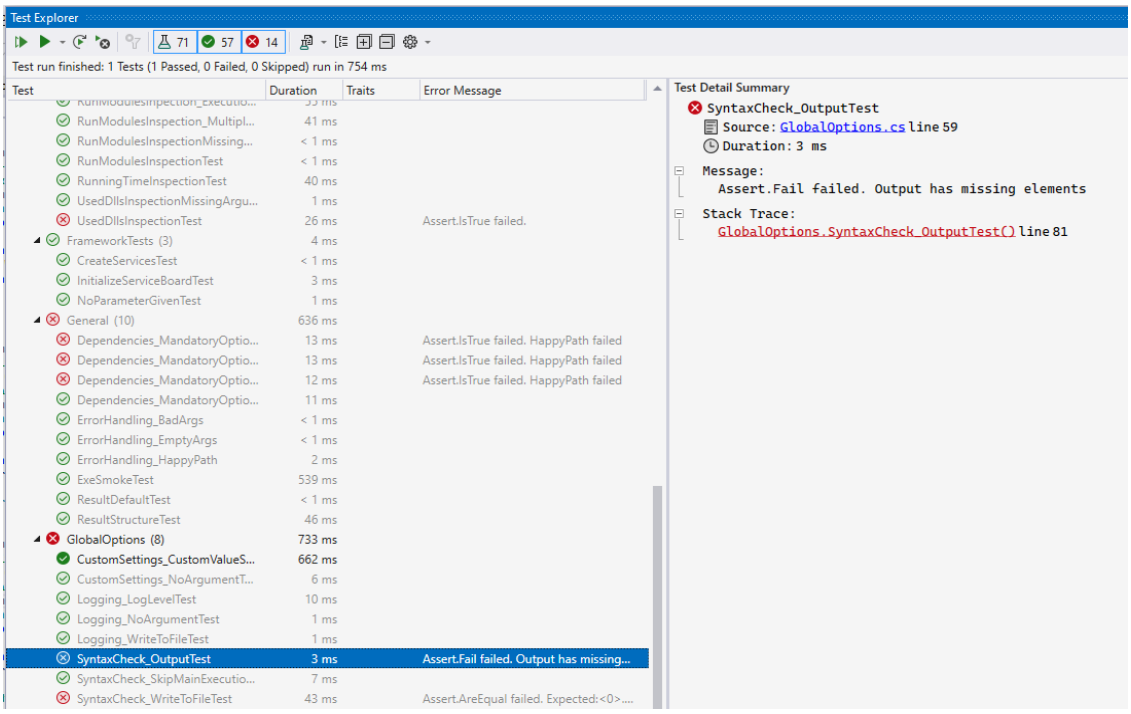
A szabály értelmében nem szabad olyan tesztet commit-olni, ami hibára fut. Kivételt képez ez alól minden olyan hiba, ami a szoftver funkcionális, és nem-funkcionális hibájából adódik.

A futási eredményem a [7.11 ábra]-n látható. A tesztek struktúráját a követelmények struktúrája szabja meg. Logikai szétbontásuk azon alapul, hogy a funkciókat a követelmények egy bizonyos szempont alapján csoportosítják. A correction-be tartozó funkciók tesztelése elkülönül a framework, az inspection, és általános funkciók tesztelésétől. Ennek oka, hogy az automatizált tesztek futási eredményei áttekinthetőbbek legyenek, illetve a tesztek futtatása közben ne kapjunk hamis hibákat.



7.11 ábra: Automatizált tesztek futási eredménye

A hibás tesztek hibaüzenetei nem minden esetben definiálják pontosan a hiba okát, de kellő információt tartalmaznak, ami alapján a hiba megtalálható, és kijavítható. A hiba definiálására a [7.12 ábra]-n látható egy példa:



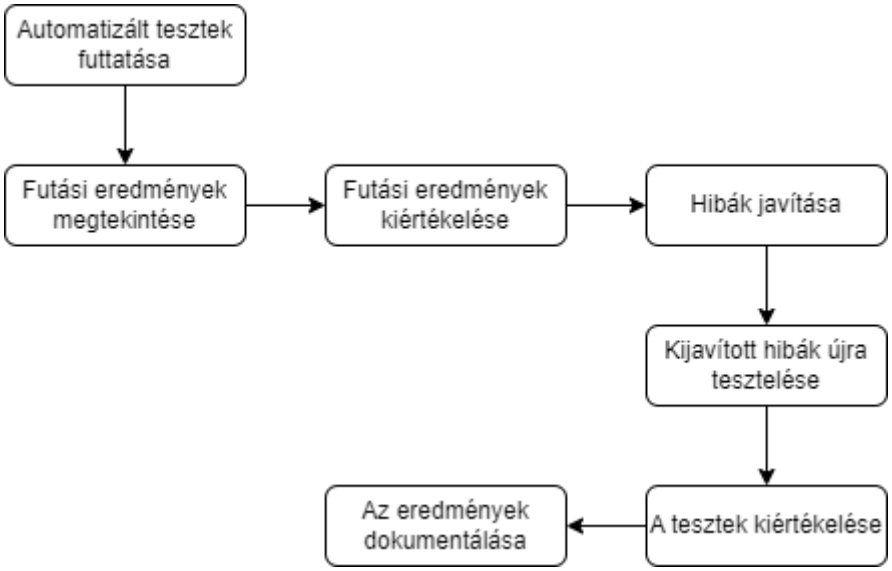
7.12 ábra: Automatizált teszt hibaüzenete

A jó eredménnyel lefutott automatizált tesztekkel nincsen további teendő a dokumentáción kívül.

7.4.2 Az automatizált tesztek eredményeinek dokumentálása és folyamata

Az automatizált tesztek futási eredményeinek dokumentálása kifejezetten fontos tényezője a tesztelés lezárásának. A folyamat alapjait az ISTQB [1][4] szabvány alapján alkottam meg.

A folyamatot az általam draw.io-ban elkészített [7.13 ábra] mutatja be:



7.13 ábra: Az automatizált tesztek futási eredményeinek dokumentálási folyamat ábrája

Az első futási eredmények megtekintése ad egy átfogó képet a technikai követelmények megfelelő működéséről. Abban az esetben, ha minden automatizált teszt hibás eredményre fut, annak technikai okai vannak. Ilyen ok lehet a környezet nem megfelelő beállítása, vagy akár a teszt dokumentumok hiánya. A beállítás megjavítását, és a teszt dokumentumok pótlását követően a tesztek nem futhatnak hibára technikai okok miatt.

A futási eredmények kiértékelése alatt a hiba üzeneteket vizsgálom meg. A hibák okai a következők lehetnek:

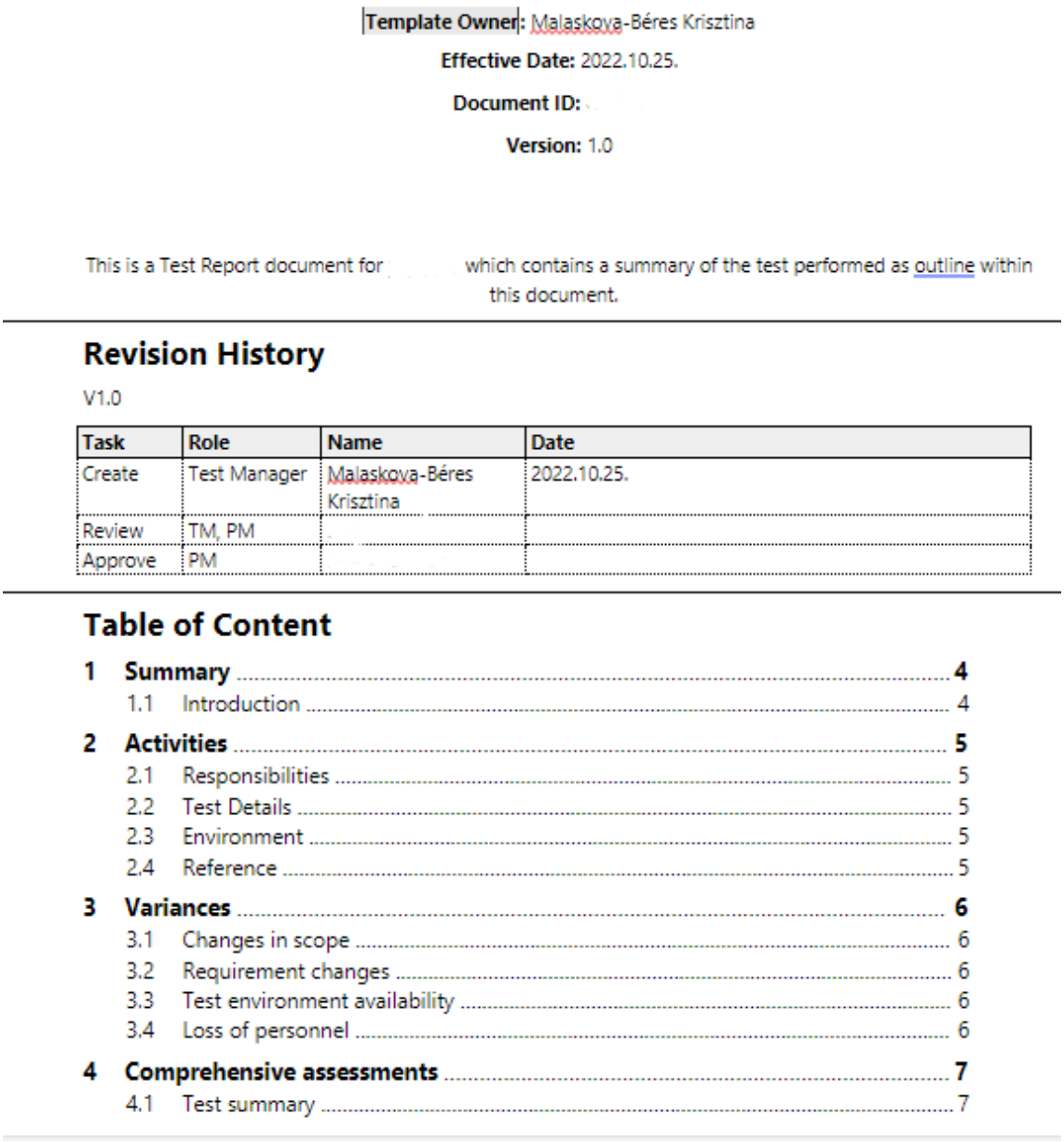
1. A lefejlesztett funkciók nem a követelményeknek megfelelően működnek.
2. A lefejlesztett funkciók megváltoztak a követelménnyel együtt, de az automatizált teszt nem lett ennek megfelelően módosítva.
3. Az szoftver elvárt válasz a tesztre időtullépés miatt meghiúsul.
4. Az automatizált teszt hibásan van megírva.

5. A követelmény rosszul lett definiálva, emiatt a fejlesztés hibásan lett megvalósítva a teszteléssel együtt.

A kiértékelést követően a hibákat ki kell javítani. A javítás történhet a fejlesztő által, vagy a tesztelő által, annak függvényében, hogy melyik kódban van a hiba. Amennyiben a hiba a követelményben van jelen, úgy a követelmény megírásáért felelős személynek módosítania kell a követelményt, és eszerint szükséges az automatizált tesztek és forráskódok módosítása is.

A hiba kijavítását követően az automatizált teszteket újból lefuttatom, és az eredményeket ismételten kiértékelem. A tévesen hibás eredményeket a tesztelők kezelik, mivel ez jellemzően olyan esetekből adódik amikért a követelmények, a fejlesztés, és az automatizált tesztek nem tehetőek felelőssé.

A valós hibák dokumentálása egy Teszt Jelentésbe kerül bele, amit én hozok létre a vállalati sablonból. A dokumentum tartalmazza a létrehozás dátumát ami a teszt befejezésének napja, a dokumentum azonosítóját, és a dokumentum verzió számát, ami a [7.12 ábra]-n látható. A dokumentum egy vállalati sablon, aminek továbbfejlesztését én valósítottam meg.



Template Owner: Malaskova-Béres Krisztina

Effective Date: 2022.10.25.

Document ID:

Version: 1.0

This is a Test Report document for which contains a summary of the test performed as outline within this document.

Revision History

V1.0

Task	Role	Name	Date
Create	Test Manager	Malaskova-Béres Krisztina	2022.10.25.
Review	TM, PM		
Approve	PM		

Table of Content

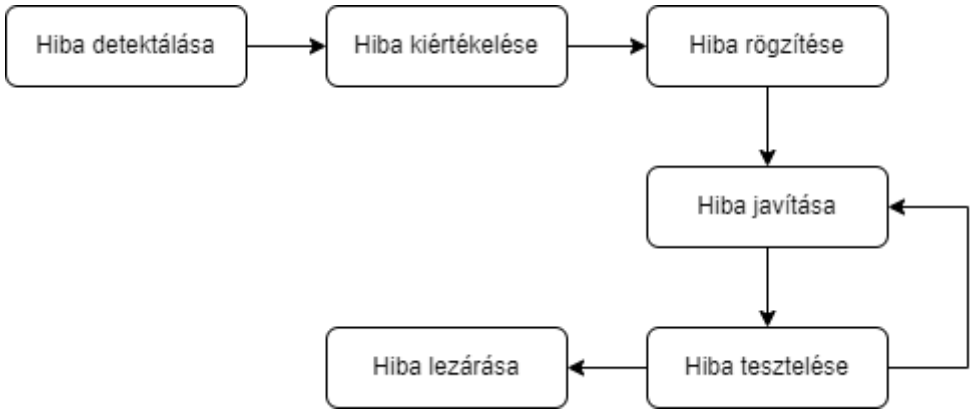
7.14 ábra: A tesztelési jegyzőkönyv dokumentumának kezdő oldala

A teszt jelentés további fontos információkat is tartalmaz. Tartalmazza az:

1. Összefoglalót a tesztelésről,
2. A tesztelés alatt meghatározott felelősségi köröket,
3. Referenciákat,
4. Teszt környezet beállításait
5. Használt alkalmazások verzió számait
6. A tesztelők neveit akik a teszteket futtatják
7. A menedzser nevét aki a teszt futtatását felügyeli

8. A tesztelés közben felmerült ember hiányt
9. Egyéb tesztelést hártáltató tényeket
10. A lefuttatott teszteket, és azok számát
11. A lefuttatott tesztek eredményeit
12. A teszt futtatása során talált hibákat
13. A talált hibák állapotát,
14. A végső összefoglalót a tesztelésről.

A teszt jelentést követően a hibák nyomonkövetése a feladatom. Számon kell tartanom, hogy az automatizált tesztelés közben talált hibák javítása hogyan fog teljesülni. A hiba életciklusát a draw.io-ban [5] általam elkésztített [7.15 ábra] mutatja be:



7.15 ábra: A hiba életciklusa

A hiba detektálását, és kiértékelését követően a hibát rögzítem a vállalat jegykezelő rendszerében, ahol az alábbi információkat adom meg:

1. A hiba súlyosságának mértéke – prioritása
2. A hibához kapcsolódó követelmény
3. A hibát detektáló automatizált teszt eset
4. A környezet adatai, verzió száma
5. Az automatizált teszt környezetének adatai
6. A szoftver verziószáma
7. A hiba előidézéséhez szükséges környezeti beállítások
8. A hiba javításának felelőse

A hiba rögzítését követően megkezdődik a hiba javítása a termék felelős, a fejlesztő, vagy tesztelő által. A javítást követően a hibát újra kell ellenőrizni. Az ellenőrzés történhet manuálisan, és automatizált teszteléssel is. és amennyiben az újra ellenőrzés sikeres eredménnyel zárul, a hiba lezárható.

A hibák jegyzőkönyvbe vételénél három szempont szerint csoportosítom a hibákat:

1. Új hibák

2. Nyitott hibák
3. Lezárt hibák

Az új hibák azok a hibák, melyek újonnan kerültek rögzítésre, és nincs olyan felelőse, aki elkezdett dolgozni a javításán. A rendszerben amit kialakítottam, egy új hibának a kezelése maximum egy héten belül meg kell valósuljon prioritás függvényében. Amennyiben a hiba alacsony prioritású, úgy nem szükséges az egy héten belüli javítása, ellenben a magas prioritású hibák esetében a javításnak egy héten belül meg kell valósulnia.

A nyitott hibák azok a hibák, melyek javítása már elkezdődött, de még nem került lezárásra. Ez lehet akkor, mikor a hibáért felelős személy még dolgozik a hiba megoldásán, és lehet akkor is, mikor a tesztelés még folyamatban van, ahol az arra megfelelő személy ellenőrzi, hogy a hiba jelen van, vagy már megjavult.

A lezárt hibák azok a hibák, amikkel nincsen további teendő. Három esetben fordulhat elő, hogy a hiba lezárásra került:

1. A hiba javult, a tesztelésen sikeresen átment.
2. A hiba már nem releváns.
3. A hiba javítása nem lehetséges, az a rendszer limitációjának része.

A hiba irrelevanciája adódhat abból, hogy a követelmény idő közben megváltozott, és a megváltoztatás kiküszöbölte a hiba jelenségét, miután a fejlesztő is ennek függvényében módosította a forráskódot. Továbbá adódhat abból is, hogy alapos utánajárást követően megbizonyosodunk arról, hogy a hiba nem a kódban van jelen, és nem is a követelményben. Ezt okozhatja a követelmény megfogalmazásából adódó félre értés is. Ebben az esetben további követelmény pontosítás javasolt, azonban nem kötelező.

A hibák lezárásának okát a hiba jelentésben nem tüntetem fel, ott csak a lezárás meglétét dokumentálom.

8. Továbbfejlesztési lehetőségek

A tesztelési folyamatok jelenleg is fejlesztés alatt állnak. Ennek kidolgozása az én feladatom. Több területen is lehetőség van a tesztelés hatékonyságának növelésére. A könyvelői rendszer tesztelői folyamatainak kidolgozása megfelelő. Bevezetésre fog kerülni a tesztelés automatizálása. Az eszközök még nincsenek meghatározva, de a tervek már elkészültek. Több lehetőség közül is lehet választani.

Egyik lehetőség, hogy a teszt automatizálás fejlesztését egy kisebb csapat fogja végrehajtani. A munkafolyamat hasonló lesz mint a manuális tesztelés esetében, annyi különbséggel, hogy az automatizált teszteléshez szükséges rendszer kiépítése több hónapot is igénybe vehet. Nem csak a tesztek lefejlesztése lesz a feladat, hanem a tesztelési módszerek kialakítása, és olyan döntések meghozatala, ami meghatározza, hogy milyen típusú tesztelést szeretnénk végrehajtani.

Úgy vélem a legfontosabb tesztelés, amit szükséges megvalósítani, az integrációs tesztelés, aminek keretein belül az adatbázisok tesztelése elengedhetetlen. Az integrációs teszteléseket olyan tesztelők kell elvégezzék, akik rendelkeznek backend tesztelő tapasztalattal, hiszen nem minden esetben van lehetőség frontend tesztelést végrehajtani.

Az adatbázisok tesztelése jelentős kihívást jelent, de továbbfejlesztés szempontjából szükséges ebbe az irányba is elindulni. A könyvelői rendszer minden funkciója a mögötte álló adatbázisra épül. Szükséges lecsökkenteni azt a kockázati tényezőt, miszerint ha adatvesztés állna be, akkor a számlázás, és szerződések kezelése nem elvégezhető. Az adatok szenzivitása, és sérülékenysége érdekében különös odafigyelést igényel a tesztelés megléte, és elvégzése.

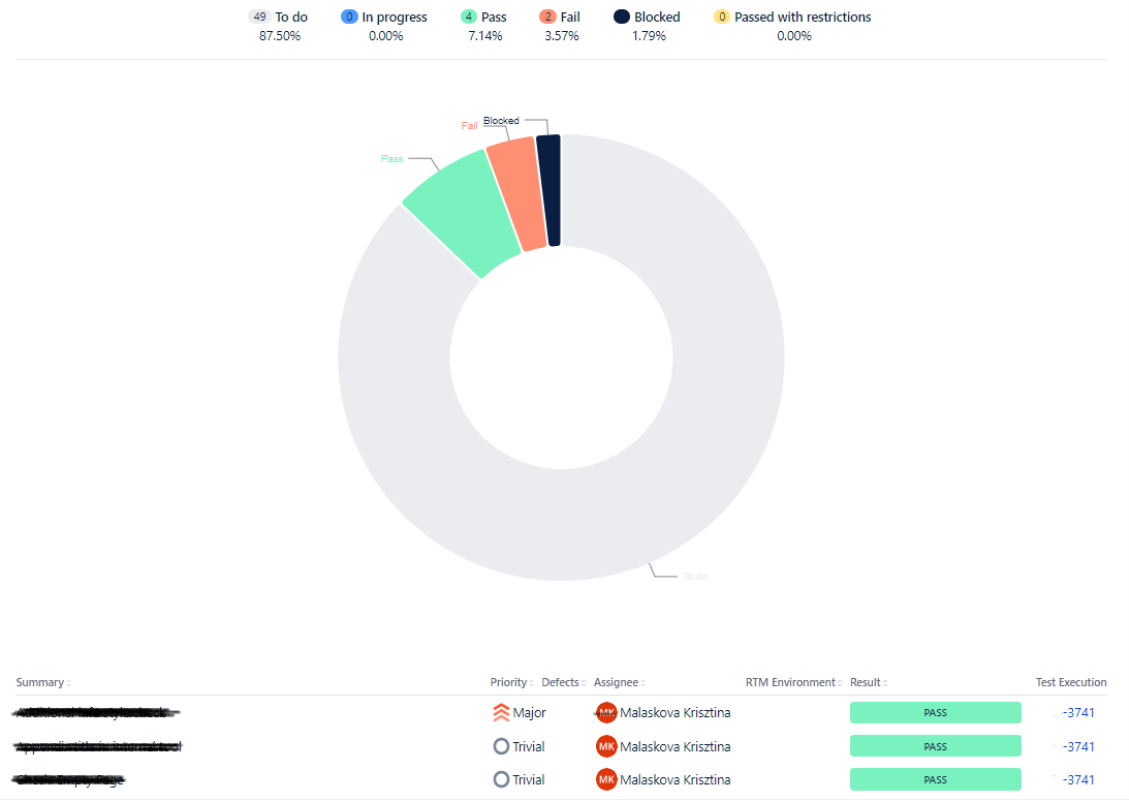
A rendszer teszteléséhez hozzá tartozik az alapos jogosultság ellenőrzés. A jelenlegi tesztelés csak korlátozott számú jogosultságot vizsgál meg, és ez olyan jellegű hibákat eredményezhet, ami a vállalat számára jelentős anyagi kárt okozhat.

Erre való tekintettel a biztonsági tesztelés bevezetése is elengedhetetlen feladat, ahol a rendszer sérülékenységét, és adatok biztonságát szükséges megvizsgálni. Ehhez szintén több rétegű munkafolyamat kialakítása, és megvalósítása szükséges.

A tesztelési eredmények jelentése, és dokumentálása is továbbfejlesztendő feladat. A vezetőség számára vizuális diagrammok létrehozása lenne célszerű, ahol nyomonkövethető a tesztelési folyamatok aktuális állapota, illetve a tesztelés végső eredménye. Ilyen jellegű diagrammok létrehozására a jegykezelő alkalmazásban van lehetőség, ami hasonlóan nézne ki, mint a [8.1 ábra]-n. Az alkalmazás, amivel a diagrammot létrehoztam a Jira [18] egyik kiegészítő modulja, az RTM. [19]

A Jira egy olyan Management Tool, ahol lehetőség van számos szoftverfejlesztéshez szükséges eszköz kezelésére, és használatára. Ilyen eszközök többek között a követelmények dokumentációja, és azok életciklusának dokumentálása, nyomon követése, a hozzájuk kapcsolódó fejlesztési feladatok létrehozása, teszt esetek összefűzése a követelményekkel, és a teszt esetek lépéseinek definiálása, futtatása, és azok eredményeinek tárolása.

Az RTM modul a Jira-ba telepíthető applikáció. Teljes nevén „Requirements & Test Management for Jira”[19]. „A Requirements and Test Management for Jira Cloud a teljes szoftverprojektet a követelményektől a gyártásig az Atlassian csomagon belülre helyezi. A Jira natív funkcióira épül, amelyek segítenek a tesztelés zökkenőmentes integrálásában kiegészítve a követelményekkel, a fejlesztéssel és a jelentéskészítéssel.”[19]



8.1 ábra: Jira RTM-ben létrehozott diagram

Az RTM modul használatával és alkalmazásával a felsővezetés számára készített jelentések elkészítése a további feladatok egyike.

Az automatizált tesztek továbbfejlesztéséhez szükséges a kódlefedettség meghatározása. Ahhoz, hogy a kódlefedettséget meg tudjam határozni, szükség van a unit tesztek megírására. Erre jelenleg nincsen lehetőség az erőforrás hiánya miatt, de a tesztelésen belül egy fontos eleme a jövőben meghatározott céloknak.

Fontos még kiemelnem az automatizált tesztek kommentjeinek fejlesztését, illetve a lefejlesztett automatizált tesztek dokumentált review-ját. A kommentek részletessége fontos szempont a tesztek továbbfejlesztésében, mivel a belső folyamatok miatt nem minden esetben ugyan az a személy fejleszti az automatizált teszteket, továbbá a tesztek alapos dokumentációja a kommentek hiányában szinte lehetetlen. Ezért szükség van a munkafolyamatok további átalakítására.

A review dokumentumot korábban elkészítettem és bemutattam, a munkafolyamatba való bevezetése még hátra van.

Ennek a review módszernek köszönhetően a tesztelés, és minőségbiztosítás stabilabbá tehető. Továbbfejlesztési lehetőség a review eseményének beépítése a munkafolyamatba, és a munkatársak betanítása a review alkalmazására. Ez vonatkozik az automatizált tesztek fejlesztésére, és a manulis tesztek megírására is.

A review folyamatok részletes leírását az ISTQB [4] szabvány tartalmazza, ahol bemutatásra kerül az összes review típus, azok használata, és alkalmazása. Ennek bevezetését a jövő évben tervezzük vállalati szinten.

Számos egyéb továbbfejlesztési lehetőség áll még rendelkezésre, de mindegyik elfogadását a vállalat vezetősége hagyja jóvá.

9. Összefoglalás

A feladatom, és a szakdolgozatom célja az volt, hogy tervezzek és készítsek egy olyan rendszert, ami egy adott vállalaton belül gördülékenyebbé teszi a tesztelést, és bevezeti a tesztautomatizálást.

Első lépésként megvizsgáltam az adott vállalat jelenlegi munkafolyamatait, és felmértem annak hatékonyságát. A vállalaton belül több vezetővel, és alkalmazottal is beszéltem, hogy megfelelő információ birtokában tudjak fejlesztési javaslatot kidolgozni.

Úgy találtam, hogy a tesztelés kiépítését minimális átalakítással lehetőség van jelentősen hatékonyabbá tenni, ezért úgy módosítottam a munkafolyamatot, hogy az minden résztvevő számára megfelelő legyen, és a vállalatnak ne kerüljön plusz költségébe, ezért egy olyan folyamattal egészítettem ki a tesztelést, hogy az egyben a minőségbiztosítás hatékonyságát is növelje. A tesztelés teljes mértékben átkerült a tesztelői csapathoz, és az üzleti szereplők tehermentesítve lettek a teszteléstől.

A teszt rendszerek átalakítását is el kellett végeznem ahhoz, hogy a tesztelés hatékonyságát, és stabilitását növelni tudjam. A meglévő teszt környezetek struktúráját átalakítottam úgy, hogy a teszt környezet elemei megmaradtak, és a biztonsági mentést is meg lehetett valósítani, így nem áll fent a veszélye az adatvesztésnek.

Az automatizált tesztek bevezetése szintén megvalósult, ahol az alap termék továbbfejlesztett verzióján már a kezdetektől a teszt automatizálás lett alkalmazva mint tesztelés. Így lehetőség adódott arra, hogy a tesztelői erőforrással gazdálkodni tudjak, és a vállalat számára nincsen szükség további alkalmazottak felvételére, így a tesztelés automatizálása költséghatékony módszer is lett.

Az automatizált tesztek fejlesztéséhez szükséges eszközöket nem volt szükség kiválasztani, mivel vezetői döntés született arról, hogy milyen alkalmazásban kell az automatizált tesztek fejlesztését, és futtatását végrehajtani.

A szakdolgozat létrehozása alatt segítségett nyújtott számomra az ISTQB tesztelői szabványban leírt metodológiák ismerete, és alkalmazása. A már meglévő folyamatok kiegészítése kapcsán volt leginkább irányadó az ISTQB, és a továbbfejlesztési lehetőségek áttekintéséhez nyújtott segítséget.

A szakdolgozatban ismertett, és lefejlesztett folyamatok, és automatizált tesztek a vállalatnál ma már teljes mértékben használatba léptek, és alkalmazzák őket. Tekintve a kezdeti célt, a feladatot sikeresen teljesítettem, és a vállalat szereplői is elégedettek a fejlesztésekkel, mert az a mindennapi munkájukat könnyíti meg. Sikerült cégen belül olyan további víziót kialakítanom a továbbfejlesztési tevekkel, ami a tesztelés hatékonyságának növelésében fog segítséget nyújtani a jövőben.

10.Summary

My task, and the aim of my thesis, was to design and build a system that would make testing and test automation smoother within a company.

As a first step, I examined the current workflow of the company and assessed its efficiency. I also talked to several managers and employees within the company, so that I could make recommendations for improvement with the right information.

I found that the way testing is set up could be made significantly more efficient with minimal changes, so I modified the workflow to suit all participants and at no extra cost to the company, and added a process to the testing that would also increase the efficiency of quality assurance. Testing was fully transferred to the testing team and business stakeholders were relieved of the burden of testing.

I also had to redesign the test systems to increase the efficiency and stability of testing. I restructured the existing test environments so that the elements of the test environment were preserved and backups could be implemented so that there was no risk of data loss.

The introduction of automated tests has also been implemented, where test automation has been used as a test from the beginning on the enhanced version of the base product. This enabled the company to manage tester resources and avoid the need to hire additional staff, making test automation a cost-effective method.

The tools for developing automated tests did not need to be selected, as a management decision was made on the application in which the automated tests should be developed and run.

During the creation of this thesis, I was assisted by my knowledge and application of the methodologies described in the ISTQB Tester Standard. The ISTQB has been most helpful in supplementing existing processes, and has provided guidance in reviewing opportunities for further development.

The processes and automated tests described and developed in the thesis are now fully used and applied in the company. Considering the initial objective, the assignment was successfully completed and the company's stakeholders are satisfied with the improvements, as it facilitates their daily work. I have succeeded in creating an additional vision within the company with further development camels that will help to increase the efficiency of testing in the future.

Irodalomjegyzék

[1]Certified Tester Foundation Level Syllabus (https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html), utoljára megtekintve: 2022. 05. 08.

[2]Automotive Spice – Process Reference Model, Process Assessment Model (http://www.automotivespice.com/fileadmin/softwaredownload/Automotive_SPICE_PAM_30.pdf), utoljára megtekintve: 2022. 11. 29.

[3]Microsoft Word (https://docs.microsoft.com/en-us/office/client-developer/word/word-home), utoljára megtekintve: 2022. 09. 08.

[4]ISTQB CTFL (https://www.hstqb.org/en/downloadarea/istqb-ctfl-syllabus-2018-v3-1-magyar/), utoljára megtekintve: 2022. 12. 11.

[5]Draw.io – alkalmazás használata az ábrák elkészítéséhez (https://app.diagrams.net/), utoljára megtekintve: 2022.12.04.

[6]Black box/Feketedoboz tesztelés – fogalma az ISTQB szerint (https://istqb-glossary.page/hu/feketedoboz-tesztelesi-technika/), utoljára megtekintve: 2022.11.17.

[7]Nem funkcionális tesztelés – fogalma az ISTQB szerint (https://istqb-glossary.page/hu/feketedoboz-tesztelesi-technika/), utoljára megtekintve: 2022.11.17.

[8]Funkcionális tesztelés – fogalma (http://users.atw.hu/kirschadam/teszteles/Szoftvertesztel%C3%A9s%20elm%C3%A9let.html), utoljára megtekintve: 2022.11.17.

[9]Nem funkcionális tesztelés – fogalma (http://users.atw.hu/kirschadam/teszteles/Szoftvertesztel%C3%A9s%20elm%C3%A9let.html), utoljára megtekintve: 2022.11.17.

[10]Automatizált tesztelés – fogalma (http://users.atw.hu/kirschadam/teszteles/Szoftvertesztel%C3%A9s%20elm%C3%A9let.html#automata_teszteles), utoljára megtekintve: 2022.11.20.

[11]Integrációs tesztelés – fogalma
(http://users.atw.hu/kirschadam/teszteles/Szoftvertesztel%C3%A9s%20elm%C3%A9let.html#integracios_teszt), utoljára megtekintve: 2022.11.20.

[12]Rendszerteszt – fogalma
(http://users.atw.hu/kirschadam/teszteles/Szoftvertesztel%C3%A9s%20elm%C3%A9let.html#rendszerteszt), utoljára megtekintve: 2022.11.20.

[13]Elfogadási teszt – fogalma
(http://users.atw.hu/kirschadam/teszteles/Szoftvertesztel%C3%A9s%20elm%C3%A9let.html#elfogadasi_teszt), utoljára megtekintve: 2022.11.20.

[14]Microsoft Visual Studio 2022
(https://visualstudio.microsoft.com/vs/), utoljára megtekintve: 2022.11.20.

[15]C# programozási nyelv
(https://learn.microsoft.com/en-us/dotnet/csharp/), utoljára megtekintve: 2022.11.20.

[16]MSTest Test Project (.NET Core)
(https://learn.microsoft.com/en-us/dotnet/csharp/), utoljára megtekintve: 2022.11.22

[17]Mi a Mock teszt – ennek fogalma angolról magyarra fordítva
(https://www.telerik.com/blogs/how-to-simplify-your-csharp-unit-testing-mocking-framework), utoljára megtekintve: 2022.12.06

[18]Jira Management Tool
(https://www.atlassian.com/software/jira/guides/getting-started/overview), utoljára megtekintve: 2022.12.10

[19]Jira RTM module
(https://marketplace.atlassian.com/apps/1220294/requirements-test-management-for-jira?tab=overview&hosting=cloud), utoljára megtekintve: 2022.12.10