Szakdolgozat

Miskolci Egyetem

Tesztelési módok szerepe és jelentősége a szoftverfejlesztési folyamatokban

Készítette:

Orosz Dénes Milán

Programtervező informatikus

Témavezető:

Piller Imre

Konzulens:

Albert Laura

Miskolc, 2022

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Orosz Dénes Milán (HC1Y8Y) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: Szoftvertechnológia

A szakdolgozat címe: Tesztelési módok szerepe és jelentősége a szoftverfejlesztési folyamatokban

A feladat részletezése:

A tesztelés a szoftverfejlesztési folyamat egészére nézve kiemelt jelentőséggel bír. A dolgozat azt mutatja meg, hogy mikor, milyen célból, milyen fajta tesztek készítésére és használatára van szükség.

Ehhez az alapot egy olyan alkalmazás elkészítése adja, amely igyekszik minden, tesztelés, tesztelhetőség szempontjából fontos területet lefedni, beleértve az adatbáziskezelést és a felhasználói felület megvalósítását. Ezen keresztül mutatja be a szoftvertesztelés fajtáit, lépéseit, az egyes fázisokon belül alkalmazott módszereket. Az alkalmazás elkészítéséhez Java programozási nyelv kerül felhasználásra.

Témavezető: Piller Imre (tanársegéd)

Konzulens: Albert Laura (független tesztelési szakértő)

A feladat kiadásának ideje: 2022. Miskolc

. . . . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Orosz Dénes Milán**; Neptun-kód: `HC1Y8Y` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Tesztelési módok szerepe és jelentősége a szoftverfejlesztési folyamatokban* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

Maga a tesztelés a szoftverfejlesztési folyamatnak egy nagyon fontos része. Mindenki tapasztalt már hasonlót, hogy egy program nem úgy működött, ahogy annak kellett volna. Ez a hibás működés legyen az bármennyire kis mértékű, nagy időbeli és pénzbeli veszteségeket okozhat egy cégnek, sőt akár fizikai sérüléssel is járhat a jelenlegi szituációtól függően. A szoftvertesztelés lényege, hogy minimálisra csökkentse a kockázatát annak, hogy egy szoftver működés közben meghibásodik, illetve ellehetetlenítse egy olyan szoftver kiadását, amiben ismert hiba van, miközben lehetőséget ad a szoftverminőség kiértékelésére.

Ha valaki a meghallja a szoftvertesztelés szót, akkor gyakran tesztek írására és futtatására gondol, pedig maga a szoftvertesztelés egy teljes folyamat számos lépéssel, aminek csak egy része a teszt esetek megírása és az eredmények ellenőrzése.

Hasonlóan tévhit, hogy a szoftvertesztelés, csak a felhasználók által megszabott követelmények verifikációjára fókuszál, ugyanis bármilyen meg szabott speciális követelménynek való megfelelést is tartalmaznia kell, illetve a validációt is, ami már a működési környezetben ellenőrzi az igényeket.

Bármely projekt esetén a tesztelés céljai az alábbiak lehetnek:

1. A munkatermékek, mint például a követelmények, felhasználói történetek, műszaki tervek és kód hibamegelőzés céljából történő kiértékelése

2. Annak igazolása, hogy az összes meghatározott követelmény teljesült-e

3. Annak ellenőrzése, hogy a teszt tárgya teljes mértékben implementálásra került, továbbá annak, validálása, hogy a felhasználók, illetve más érintett felek elvárásainak megfelelően működik.

4. A teszt tárgyának minőségébe vetett bizalom kiépítése

5. A meghibásodások és hibák megtalálása és ezáltal a nem megfelelő szoftverminőség kockázati szintjének csökkentése

6. Megfelelő információ biztosítása az érintett feleknek, hogy ezáltal megalapozott döntést hozhassanak különösen a teszt tárgyának minőségi szintjére vonatkozóan

7. A szerződésben foglalt, jogi vagy szabályozott követelményeknek, szabványoknak való megfelelés biztosítása, és/vagy igazolni a teszt tárgyának ezen követelményeknek, szabványoknak való megfelelőségét

A tesztelés céljai változhatnak a tesztelt komponens vagy rendszer, a tesztszint és a szoftverfejlesztési életciklusmodell kontextusától függően[1, 11. oldal].

A szoftvertesztelés számos fajtával rendelkezik. Attól függően, hogy mi tartozik a tesztelés hatókörébe, különbséget teszünk a tesztelési fajták között. A szakdolgozat célja ezeknek a fajtáknak a leírása és bemutatása tesztelési környezetben, illetve környezet nélkül, amennyiben arra van lehetőség.

2. fejezet

Koncepció

2.1. A fejezet célja

A tesztelési módszerek csoportosítására több módszer is lehetséges, viszont fontos tudni, hogy optimális esetben többet is használunk a tesztelés különböző szintjein. Én manuális, ad-hoc, szisztematikus és fekete doboz tesztelési módszereket fogok alkalmazni a programomon.
A szakdolgozat folyamán többször is fogom használni a szkript szót. Ez alatt azt a lépéssorozatot értem, amiben le van írva, hogy az adott tesztet, hogyan kell elvégezni, milyen előfeltételek vannak és mi az elvárt eredmény.

2.2. Tartalom és felépítés

Mielőtt a négy módszerre részletesen kitérnék és szakirodalmakkal, illetve kutatásokkal bővíteném ki, szükségesnek érzem ezeknek a magyarázatát.

- Manuális (kézi) tesztelés: A tesztelést a gép helyett egy ember végzi, aki hibákat keres akár a működésbe, akár egy teszt szkriptbe, de ilyenkor még a szubjektív hibákat is látócső alá lehet venni.
- Ad-hoc tesztelés: Ad-hoc tesztelés alatt akár a szabad tesztelést is nevezhetjük. Ez a mód igazából nem szab korlátot és utat sem mutat a tesztelő számára. Teljes mértékben a tesztelő kezében van, hogy milyen módszereket használva teszteli az adott szoftvert.
- Szisztematikus tesztelés: A szisztematikus tesztelés az ad-hoc fordítottja. Ebben a módszertanban egy előre definiált módszerrel és tesztesettel tekintünk a szoftverre és mindig pontosan azt a funkció működését ellenőrizzük amire a teszt eset íródott.
- Fekete doboz tesztelés: Fekete doboz tesztelés alatt egy olyan módszertant értünk, ahol magát a szoftver belső működését vagy nem ismerjük, vagy ignoráljuk, ugyanis nekünk csak az számít, hogy egy adott bemenetre adott kimenetet kapjunk.

2.3. Tesztelési módszertanok és módszerek

A szoftvertesztelés módszertanaival már nagyon sok ember foglalkozott, bár többnyire az elkészült irodalmak egy-egy módszertant említenek specifikusan, vagy talán kettő ellentmondását vagy az együtt használás előnyeit mutatják be. Fontos tudni, hogy minél több módszer kerül használatra egy szoftver tesztelése közbe, annál nagyobb lefedettséget biztosít és annál kisebb lesz az esély arra, hogy hiba marad a programunkba.

2.3.1. Manuális tesztelés

A manuális tesztelést tekinthetjük a módszertanok közül a legegyszerűbbnek. Ha valaki a teszteléssel szeretne foglalkozni, mindig ezzel fogja kezdeni és nem feltételezi azt sem a szakma, hogy IT végzettséggel rendelkezzen valaki, elég ha van hozzá affinitása. Természetesen ha valakinek már végzettsége van, az nagyon megkönnyíti a munkát, de így sem elvárható, hogy rögtön egy hét után már önállóan hiba nélkül tudjon dolgozni a tesztelő. A legegyszerűbb módszertannak neveztem az előbb, mégis tudni kell hogy a legszínesebb is tud egyszerre lenni, ugyanis cégenként változik a bevált módszer és a feladatkör.
Az esetek többségében a folyamat úgy zajlik, hogy a tesztelő megkapja a tesztforgatókönyvet a projekt- vagy tesztkoordinátortól, szenior tesztelőtől, amelyben tételesen le van írva, miket kell vizsgálnia, mi az elvárt eredmény, és amennyiben nem a várt eredményt kapja, akkor mi a teendő. Ez lehet olyan triviális apróság, mint például hogy funkcionális tesztelésnél az adott gombra kattintva nem történik semmi, vagy lefagy a rendszer, nem a megfelelő oldal nyílik meg stb. A tesztelő munkája nem attól izgalmas, hogy nem talál hibát, hanem attól, ha minél többet fedez fel[2].
A szakma és a módszertan legizgalmasabb része maga a hiba felderítése és nyomozása.

- Mi az ami a hibát okozza?
- Hogy lehet reprodukálni?
- A kód melyik részében található?

Ezeknek a kérdéseknek a megválaszolásával jutunk el oda, hogy a manuális tesztelő megírja a riportot a hibáról és továbbítja a fejlesztőknek.
Ehhez viszont az adott szoftvert, annak funkcióit alaposan ismerni kell (az újat és a régit egyaránt). Ezért a junior manuális tesztelők az első 2-6 hónapban rengeteg dokumentációt olvasnak (ha van), munkájuk nagy részét felületi tesztek teszik ki, hogy minél jobban meg tudják ismerni a rendszereket. Akinek már van elég tapasztalata, annak a napi munka részét képezi a bug-tracking (a felfedezett hiba nyomon követése), a tesztmódszertan kidolgozása, a tesztesetek (újra)definiálása, a tesztforgatókönyv-készítés és a log elemzés is[2].

2.3.2. Ad-hoc tesztelés

A manuális tesztelés alatt található módszertanok közül a kevésbé használt, de nem kevésbé hasznos módszertan az ad-hoc tesztelés. Sajnos előfordulhat minden szoftver fejlesztése közben, hogy az előre megírt szkriptek nem fedik le a teljes funkciót, illetve minden apró részletet, mégis ott olyan hiba található, ami a teljes szoftvert működésképtelenné tenné.

Pontosan ilyen esetek miatt hasznos az ad-hoc tesztelés, ugyanis a maga szabálytalansága miatt, ezeket az eseteket vizsgálni tudja, viszont az így talált hibák dokumentálása és a hiba reprodukálása nehezebb feladat tud lenni, ugyanis nincs mihez viszonyítani, így ezt a fajta tesztelést a már tapasztaltabb manuális tesztelők szokták végezni. Három nagy módszer található a módszertanba.

- Buddy testing : Két kolléga kölcsönösen dolgozik ugyanazon modul hibáinak azonosításán. Többnyire az egyik a fejlesztői csapatból, a másik pedig a tesztelői csapatból érkezik. A haver-tesztelés segít a tesztelőknek jobb teszteseteket kidolgozni, és a fejlesztőcsapat is idejekorán tud tervmódosításokat végrehajtani. Ez a tesztelés általában az egységtesztelés befejezése után történik[3].

- Pair testing (Páros tesztelés) : Két tesztelő kap modulokat, megosztják egymással az ötleteiket, és ugyanazon a gépen dolgoznak a hibák felkutatásán. Az egyik személy elvégezheti a teszteket, a másik pedig jegyzetelheti a megállapításokat. A személyek szerepe lehet tesztelő és firkász a tesztelés során[3].

- Monkey testing : A célja, hogy a tesztelő véletlenszerűen tesztelje a terméket vagy alkalmazást tesztesetek nélkül, azzal a céllal, hogy tönkretegye a rendszert[3].

2.3.3. Szisztematikus tesztelés

A szisztematikus tesztelés egy nagyon széles körű módszertan, ami minden módszert magába foglal, amit az ad-hoc nem. A kettő módszertan együtt tartalmazza az összes módszert amit manuálisan alkalmazni lehet. Minden olyan módszer ami rendelkezik előre megírt szkripttel az ebbe a kategóriába sorolható. Fontos tudni, hogy az automata tesztelés csak és kimondottan szisztematikus tesztelésnél fordulhat elő, viszont a szisztematikus tesztelés az manuális tesztelés esetében is nagyon gyakori. Ide sorolható az összes fekete és fehér dobozos tesztelési technika. A fekete dobozos technikákra rögtön kitérek a következő szekcióban a fehér dobozosak pedig a következők:

- Utasítástesztelés és - lefedettség

- Döntési tesztelés és lefedettség

- Az utasítástesztelés és a döntési tesztelés értéke

Említés szintjén a fehér dobozos tesztelés a 2.4-es pontban megtalálható. A következő fejezetben részletesebben kitérek arra, milyen szerepe van a tesztelésben és a tervezésben a szisztematikus tesztelésnek.

2.3.4. Fekete doboz tesztelés

A fekete doboz teszttechnikák (viselkedési vagy viselkedés alapú technikáknak is nevezzük) a megfelelő tesztbázis (pl. formális követelmény dokumentumok, specifikációk, használati esetek, felhasználói történetek vagy üzleti folyamatok) elemzésén alapulnak. Ezen technikák mind a funkcionális, mind a nem funkcionális teszteléshez alkalmasak. A fekete doboz teszttechnikák a tesztelés tárgyának bemeneteire és kimeneteire koncentrálnak, a belső szerkezetre történő hivatkozás nélkül.
Ez a módszertan 5 teszttechnikát tartalmaz:

- Ekvivalencia particionálás : Ezt a technikát használva, partíciókra osztjuk a letesztelendő feladatokat, és értékeket határozunk meg amik vagy negatívak vagy pozitívak. Ezek alapján az elvárt eredmény változik. Fontos, hogy minden érték csak 1 partícióhoz tartozhat és 1 tesztesetben nem lehet kombinálni több negatív partíciót, ezzel kiszűrve annak az esélyét, hogy 2 hibát 1-nek gondolunk[4, 56-60. oldalak].

- Határérték-elemzés : A határérték-elemzés igazából az ekvivalenciapartíció kibővítése, viszont csak numerikus vagy szekvenciális elemeket tartalmazó részeken használható. Ennél a tesztelésnél egy intervallummal rendelkezünk, amikre pozitív elvárt eredményt kapunk és minden másra pedig negatívot. A határértékek mindig ennek az intervallumnak a legkisebb és a legnagyobb eleme[4, 56-60. oldalak].

- Döntési tábla tesztelés : Ebben a teszttechnikában táblázatot hozunk létre, aminek a sorait a rendszer követelményei és az eredményezett műveletek alkotják. Minden oszlop egy döntési szabálynak felel meg, ami a feltételek egy olyan egyedi kombinációját definiálja, ami az ehhez a szabályhoz rendelt műveletek végrehajtását eredményezi[4, 56-60. oldalak].

- Állapotátmenet tesztelés : A teszttechnika használata közben a tesztelt funkció jelenlegi állapotát, az átmeneti állapotot és a következő állapota alapján kategorizáljuk. Egy állapotból több potenciális állapot lehet például egy jelszó beírása után vagy beenged az oldal vagy nem és ezek az esetek az átmeneti állapotok. Az állapotátmeneti tábla az állapotok közötti összes érvényes és potenciálisan érvénytelen átmenetet, az eseményeket, illetve az érvényes átmenetekhez tartozó végrehajtandó műveleteket ábrázolja. Az állapotátmenet diagramok általában csak az érvényes átmeneteket mutatják és kizárják az érvényteleneket[4, 56-60. oldalak].

- Használati eset tesztelés : A tesztek származtathatóak használati esetekből is, amik a szoftverelemekkel történő kölcsönhatások tervezésének speciális módjai. Egyesítik a szoftverfunkciókra vonatkozó követelményeket. A használati eseteket aktorokhoz (emberi felhasználók, külső hardver, másik komponens vagy rendszer) és tárgyakhoz (a komponens vagy rendszer, amelyre a használati esetet alkalmazzuk) kötjük[4, 56-60. oldalak].

2.4. Amit csak említés szintjén érdemes szerepeltetni

Ebben a szekcióban nem tértem ki a fehér doboz és szürke doboz tesztelésre, illetve az automatára sem részletesen, ugyanis ezek nem kerültek alkalmazásra a program készítése és tesztelése során, de egy említést azért megérdemelnek.

- Fehér doboz tesztelés: A fehér doboz tesztelés a fekete doboz ellentéte. Ebben a módszertanban ismerjük a szoftver belső felépítését és a teszt esetek megírása közben figyelni kell arra, hogy a bemenet kielégítse az összes elágazás, belső függvény és ciklus követelményeit is.

- Szürke doboz tesztelés: A szürke doboz tesztelés a fekete és fehér doboz kombinációja. A három módszertan közül ez a legújabb és egyre több helyen használják

már. A lényegi különbség, hogy megpróbál egy arany középutat találni a két szélsőséges módszertan között felhasználva a pozitívumokat és kihagyva a negatívumokat.

- Automata tesztelés: Az automata tesztelés során a gép értékeli ki az eredményt, majd egyértelmű visszajelzést ad arról. Könnyen ismételhetőségre nyújt lehetőséget, rövid idő alatt sok esetet tud vizsgálni és ideális esetben hibázni sem hibázik, viszont nem mindig lehet alkalmazni és a tesztesetet rendkívül pontosan kell megfogalmazni.

Természetesen ez teljesen saját preferencia és egyik módszert sem merném kijelenteni, hogy jobb a másiknál, de az adott körülmények egyes módszerek használatát egyszerűbbé teszi.

3. fejezet

Tervezés

A szakdolgozatom lényege a tesztelés bemutatása és az alkalmazott technikák szemléltetése így maga a program ami készül hozzá, kimondottan ilyen céllal lett tervezve is, emiatt inkább kutatás jellegű a dolgozat. Ebben a részben a tesztelés tervezését részletezem illetve egy bővebben beszélek a szisztematikus tesztelésnek a szerepéről.

3.1. A tesztelés megtervezése

Mint a szoftverfejlesztés, maga a tesztelés is egy tervel indul el. A tesztelési terv (röviden: teszterv) összefoglalja a fejlesztés és az üzemeltetés során elvégzendő tesztelési tevékenységeket. A tervezést a következő tényezők tudják befolyásolni:

1. A szervezet tesztstratégiája
2. A tesztelés irányelvei
3. A fejlesztési ciklus
4. Az alkalmazott módszerek
5. A teszt tárgya
6. A teszt céljai
7. A teszttel járó kockázatok
8. A teszthez tartozó megkötések
9. A kritikusság
10. Az elérhető erőforrások
11. A tesztelhetőség

Egy projekt, illetve a teszterv minél későbbi fázisban van annál több információ áll rendelkezésünkre és annál részletesebb tervet tudunk készíteni. A teszttervezést egy dinamikus tevékenység, amit minden fázisban el kell végezni. A tesztevékenységekből kapott információk alapján már jobban bele lehet tekinteni a kockázatokba és esetleges módosításokat lehet végezni ez alapján a tervbe.
A tervezést dokumentálhatják a fő tesztterben és az egyes tesztszintekhez tartozó

teszttervekben, mint például a rendszertesztelés, elfogadási tesztelés, vagy a különböző teszttípusokhoz tartozó teszttervekben, mint például a használhatósági tesztelés és a teljesítménytesztelés[5, 67. oldal].
A teszttervezési tevékenységekhez tartozhatnak és dokumentálhatóak a tervben :

- A kockázatoknak, a tesztelés tárgyának és a céloknak a definiálása
- Az általános megközelítés kifejtése
- Egy nagyobb leíró rész, arról, hogy mit fogunk tesztelni, azt ki fogja tesztelni, milyen erőforrásokra van ehhez szükség és hogyan kell a tevékenységeket végrehajtani
- A tesztfelügyelethez és az irányításához a metrikák kiválasztása
- A költségvetés meghatározása
- A dokumentáció struktúrájának, illetve részletességének a meghatározása
- A teszttevékenységek koordinálása, és az életciklusba beépítése
- A terv különböző pontjai (elemzés,tervezés,megvalósítás,végrehajtás) ütemezése adott időpontokra, vagy iterációkhoz kötve.

3.2. A tesztstratégia és a tesztelési megközelítés

Maga a tesztstratégia a tesztelési folyamatot mutatja be általános megközelítéssel, többnyire egy szervezet vagy egy termék szintjén.
A következők az általános típusai a tesztstratégiáknak:

3.2.1. Analitikus

Az analitikus megközelítés olyan típusa a tesztstratégiáknak, ahol egy bizonyos tényezőt veszünk középpontba (például kockázat vagy esetleg követelmény) és ez a tényező kerül elemzésre. Kockázat alapú tesztelésnél azt jelenti az analitikus megközelítés, hogy az elkészülő teszt eseteket kockázat alapján priorizálják és tervezik meg.

3.2.2. Modell alapú

Ahogy a neve is mutatja, a teszteket néhány modell alapján készítjük el, amik általában a termék egy jellemzőjére épülnek (például üzleti folyamat, belső szerkezet). Az állapot modellek, a megbízhatósági növekedés modellek és az üzleti folyamat modellek tudnak példát adni ezekre a modellekre.

3.2.3. Módszeres

Ez a tesztstratégia a tesztek és tesztfeltételek néhány előre meghatározott halmazát használja szisztematikusan. Ilyen például a fontos minőségi jellemzők listája, vagy a vállalati szintű megjelenés, de a gyakori hibatípusok csoportjai is ide sorolhatóak.

3.2.4. Folyamat szerinti

Ez a fajta stratégia külső szabályok és szabványokon alapul. Ez azt jelenti, hogy a tesztek tervezése, elemzése és megvalósítása ezeken alapul. Ezek a szabványok vagy a szervezet által vagy a szervezetre lettek meghatározva.

3.2.5. Irányított

Ezt a stratégiát nagyban befolyásolja a technológiai szakértőktől vagy az érdekelt felektől érkező útmutatás vagy utasítás. Ezek a személyek általában a teszt csapaton, de akár a szervezeten kívüli személyek is lehetnek.

3.2.6. Regresszió-kerülő

Az ilyen típusú tesztstratégia hátterében az a vágy áll, hogy elkerüljük a meglévő képességek romlását. Ez a tesztstratégia magában foglalja a meglévő tesztverek (különösen a tesztesetek és tesztadatok) újra felhasználását, a regressziós tesztek széles körű automatizálását és a standard tesztkészleteket[6, 68].

3.2.7. Reaktív

Az előző stratégiák helyett, ez a típus a tesztelés előre eltervezése helyett, inkább a komponens vagy a rendszer lényegére és a teszt végrehajtása során bekövetkezett eseményekre reagál. A stratégia a nevét is a reagálás rész miatt kapta. Nagyon gyakran a felderítő tesztelést alkalmazzák a reaktív stratégiákban[6, 68. oldal].

4. fejezet

Tesztelés a gyakorlatban

Minden teszt esethez tartoznak különböző információk, amik változnak a cég által használt rendszertől függően. Lehetséges, hogy egy teszt esetet létre tud hozni a kreálója, minden mellék információ megadása nélkül, de általában van néhány alap feltétel is amiket az én programom keretében a 4-es táblázat mutatja be.

A teszt készítője	Orosz Dénes Milán
A teszt fajtája	Sikeres bejelentkezési teszt
A teszt elő követelményei	A program bármely futtatható verziója
A teszt stratégia verziószáma	1.0
Kapcsolat(ha van)	Sikertelen login teszt

4.1. táblázat. A teszt esethez tartozó információk.

Minden teszt eset 3 vagy 4 oszlopból áll. Az első oszlopban a lépés sorszáma, a másodikban maga a lépés van megfogalmazva, a harmadik és negyedik oszlop pedig az elvárt eredményt, illetve a lépéshez tartozó bemenetet tartalmazzák, viszont előfordulhat, hogy a bemeneteket egy másik táblázatban tárolják. Példát erre a 4.2-es táblázat mutat.

Lépés sorszáma	A tesztlépés leírása	A tesztlépéshez tartozó bemeneti érték (ha van)	A tesztlépés elvárt értéke
1.			
2.			
3.			
4.			

4.2. táblázat. Egy teszt eset teszt lépések nélkül

A 4.3-mas táblázaton látható, az én programomhoz tartozó bejelentkezést tesztelő teszt esetnek a lépéssorozata és a hozzá tartozó bemenetek, illetve az elvárt eredmények.

Lépés sorszáma	A tesztlépés leírása	A tesztlépéshez tartozó bemeneti érték (ha van)	A tesztlépés elvárt értéke
1.	Indítsd el a programot.	-	A program elindul és egy bejelentkező ablak látható a képernyő közepén.
2.	Írd be a megadott teszt adatokat.	Felhasználó név: test Jelszó: test123	A felhasználónév olvasható, viszont a jelszó helyén **** látunk.
3.	Kattints rá a "login" gombra.	-	A gomb megnyomása után sikeresen bejelentkezünk.
4.	Lépj ki a programból az x segítségével.	-	A program bezáródik.

4.3. táblázat. A programhoz tartozó sikeres bejelentkezés teszt esete

A 4.4-es táblázatban látható a program egyik fő funkciójának a tesztje, amivel a fa struktúrához lehet hozzáadni egy fő elemet, illetve ha szeretné a felhasználó akkor még plusz levél elemeket a fő elem alá, viszont a teszt ezt a részt már nem tartalmazza, mivel egy másik funkcionális teszt tartalmazza azt.

Lépés sorszáma	A tesztlépés leírása	A tesztlépéshez tartozó bemeneti érték (ha van)	A tesztlépés elvárt értéke
1.	Indítsd el a programot és jelentkezz be.	Felhasználó név: test Jelszó: test123	A program elindul és a bejelentkezés is sikeres.
2.	Kattints a "New Node" gombra.	-	3 új gomb jelenik meg ("Get Information","Add Node(s)", "Back"), illetve egy szöveget elfogadó rész, ami az új elem nevét várja értékként.
3.	Írd be a teszt adatot, majd kattints a "Get Information" gombra és utána az "Add Node(s)" gombra	Teszt adat: "érték"	A gombok megnyomása után a fa struktúrában létrejön az "érték" nevű levél és kijelölésre kerül.
4.	Lépj ki a programból az x segítségével.	-	A program bezáródik.

4.4. táblázat. A fa struktúra módosítása új elem hozzáadásával

A 4.5-ös táblázatban látható a program egy másik fő funkciójára írt teszt eset. A fában lévő elemek reprezentálják a programhoz csatolt adatbázis táblaneveit és ezen táblák adatait lehet kiíratni egy kattintással. Ha duplán kattintunk, akkor is csak egyszer íródnak ki az elemek, viszont lenyílásra kerül a fában a tábla és ha a most már látható levelekre kattintunk, akkor a csak hozzá tartozó elemek kerülnek kiírásra.

Lépés sorszáma	A tesztlépés leírása	A tesztlépéshez tartozó bemeneti érték (ha van)	A tesztlépés elvárt értéke
1.	Indítsd el a programot és jelentkezz be.	Felhasználó név: test Jelszó: test123	A program elindul és a bejelentkezés is sikeres.
2.	Kattints a fa struktúrában egy fő elemre.	-	Az adatok amiket tartalmaz az elem kiírásra kerülnek a program alsó részén.
3.	Dupla kattintással nyisd le ugyanezt az elemet és kattints rá az egyik alatta lévő levélre	-	Az előzőleg kiírt adatokból, most csak a levélhez tartozó rész íratódik ki a program alsó részén.
4.	Lépj ki a programból az x segítségével.	-	A program bezáródik.

4.5. táblázat. A fa struktúrából adat kinyerés

4.1. Fekete doboz tesztelési módszerek a gyakorlatban

4.1.1. Ekvivalencia particionálás

A 4.1.1-es táblázatban láthatunk egy példát az ekvivalencia particionálásra. Ez a példa a programom bejelentkező felületére hoz egy példát.

Bemenet	Érvényes	Érvénytelen
Felhasználónév	V1: test V2: admin	I1: üres I2: bármi más szó
Jelszó	V3: test123 V4: admin	I3: üres I4: bármi más szó

4.6. táblázat. Példa az érvényes és érvénytelen adatokra

Az alábbi táblázatban látható néhány lehetséges kombináció a bejelentkezéshez. Minden sorhoz tartozik egy felhasználó név és egy jelszó, illetve az adott kimenetel. Fontos, hogy ha egy olyan felhasználót vagy jelszót adunk meg, ami érvénytelen, ahhoz már másik érvénytelen adat ne kerüljön, ugyanis ott nem tisztázható, hogy melyik okozza a hibás működést.

Teszt	Felhasználónév	Jelszó	Kimenet
T1	V1	V3	Érvényes
T2	V1	V4	Érvényes
T3	V2	V3	Érvényes
T4	V2	V4	Érvényes
T5	V1	I3	Érvénytelen
T6	I1	V3	Érvénytelen
...			

4.7. táblázat. Példa az ekvivalencia particionálási táblázatra

4.1.2. Határérték-elemzés

A programon belül a határérték elemzést bármelyik részre lehet alkalmazni, ahol numerikus megfelelés alapján történik a vizsgálat. Fontos tudni, hogy akkor is használható, ha a megfeleltetésünk nem két határ érték közé tartozik, hanem mondjuk az összes pozitív szám jó a feltétel szerint. Tipikusan hibás operátorok, vagy ciklus be és kilépési pontoknál lévő hibákat találunk meg ezzel a módszerrel. Példát a következő lista tartalmaz.

- Határértékek a leadott SQL parancsok esetén
- Select
  - Megfelelő számok 0 - $+\infty$
  - Tesztelendő számok: -999, -2, -1, 0, 1, 2, 999
- Update
  - Megfelelő számok 1 - $+\infty$
  - Tesztelendő számok: -999, -1, 0, 1, 2, 999
- Delete
  - Megfelelő számok 1 - $+\infty$
  - Tesztelendő számok: -999, -1, 0, 1, 2, 999
- Alter
  - Megfelelő számok 0

  - Tesztelendő számok: -1, 0, 1

- Insert
  - Megfelelő számok 1 - $+\infty$
  - Tesztelendő számok: -999, -1, 0, 1, 2, 999

- Create
  - Megfelelő számok 0
  - Tesztelendő számok: -1, 0, 1

- Drop
  - Megfelelő számok 0
  - Tesztelendő számok: -1, 0, 1

4.1.3. Döntési tábla tesztelés

A programon belül a döntési tábla tesztelést általában egy jól körülírható funkcióra szokták alkalmazni, mint sem hogy a teljes alkalmazásra. Tételezzük fel, hogy a program a lefuttatott sql kód alapján meg akar keresni egy személyt, a táblák között és ehhez 3 paramétert vár el a kódban.

- Név
- Beosztás
- Kód

Ezeknek a paramétereknek 2-2 lehetséges bemenetük van. Vagy helyes a megadott adat, vagy helytelen. Az egyszerűség kedvéért, nem foglalkozunk azzal, hogy üres bemenetet adunk meg.

Létezik egy negyedik sor a táblához ami a kimenetet fogja nekünk adni. Ebben a szituációban ezek hibaüzenetek lesznek, mert valamelyik adat nem felel meg a kritériumoknak. A tábla a következőképpen néz ki.

Név	Beosztás	Kód	Hibaüzenet
			Helyes adat megadás
+			Nem található a név
	+		Nem található a beosztás
		+	Nem található a kód
+	+		Nem található a név és a beosztás
+		+	Nem található a név és a kód
	+	+	Nem található a beosztás és a kód
+	+	+	Nem található a név, a beosztás és a kód

Ahol "+" jel látható, az mutatja, hogy az az adat megadásra került, viszont valamilyen hibával és így nem felel meg a kritériumnak. Minden sor 1 teszt esetet fed le, ami jelen pillanatban 8, mivel 3 bemenetünk van 2-2 fajta bemenettel és ezért a sorok száma 2*2*2 = 8 lesz.

4.1.4. Állapotátmenet tesztelés

Ahogy a név is sugallja, a program állapotának változása alapján nézzük, hogy minek kellene történnie és mi az, ami valójában történik. Erre nagyon egyszerű példa tud lenni egy ATM rendszer működése, viszont a programom login rendszere is megfelel a

bemutatásra.

A rendszerbe bejelentkezéskor meg kell adni egy felhasználó nevet és egy jelszót. Ha a helyes jelszó és felhasználó név kombináció kerül megadásra, akkor a program beenged minket, egyéb esetben, újra kell próbálkoznunk. A maximális próbálkozások száma változhat és nem is lényeges abból a szempontból, hogy hány esélyünk van a próbálkozásoknál, hanem inkább a lényeges része az állapotok és az ott lezajló események.

A jelenlegi példában 3 sikertelen próbálkozás után automatikusan bezáródik a bejelentkezési felület és a program vele együtt. A következőképpen néz ki a példa táblázat formájában.

Állapot	Helyes felhasználó név	Helyes jelszó	Helytelen adat
1. próba	test admin	test123 admin	jelszó: test
2. próba	test admin	test123 admin	jelszó: test1234
3. próba	test admin	test123 admin	jelszó: admin123
A program bezárul			
A program újraindul			
1. próba	test admin	test123 admin	-
Sikeres bejelentkezés			

Ilyenkor az állapotok között lehet változás. Általában a megadott helytelen adatra specializált visszajelzést kap a felhasználó. Példaként a következő táblázat szolgál.

Hiba indoka	Hibaüzenet
Helytelen felhasználó név	A felhasználó név nem megfelelő
Helytelen jelszó	A jelszó nem megfelelő
Helytelen felhasználó és jelszó	A megadott adatok nem megfelelőek

4.1.5. Használati eset tesztelés

Talán a legösszetettebb teszt esetek a használati eset tesztelésnél jönnek létre. Általában több egység kombinált működését teszteljük, ami elég hosszadalmas eseteket tudnak létrehozni.

Ha az alkalmazásnak van 1 fő funkciója, ami mellé tartalmaz még néhány más, de kevésbé fontos részt a működés szempontjából, akkor 1 fő ágat hozunk létre, ami a fő funkciót teszteli, majd ezután térünk rá a mellék ágakra.

Mostanában azért elég ritka az ilyen fajta alkalmazás, inkább több szétterjedő ágról beszélhetünk. Ilyenkor több teljes ágakat tesztelő részt készítünk, és utána nézzük meg a kevésbé fontos funkciókat.

Fontos tudni, hogy ilyenkor a megrendelő által megadott feltételeket teszteljük, ami elképzelhető, hogy csak a következő lesz: "Hozhassak létre egy új táblát az adatbázisban adatokkal, majd a fa struktúrában ezt el tudjam érni."

Ilyenkor ez az ő szemében csak néhány kattintás, viszont tesztelési szempontból egy nagyon sok funkciónak az összetett munkáját jelenti és ezért úgy kell megírni a teszt esetet is, hogy minden használt funkciónál ellenőrzésre kerüljön először a fő rész, és utána a mellék ágak is.

Ugyanennél a példánál maradva a következőképpen néznek ki a funkciók:

- Bejelentkezés
- SQL kód megírása / bemásolása
- A tábla feltöltése adatokkal
- A fához az új tábla hozzáadása (beleértve a leveleket is)
- A fában lévő új elem ellenőrzése

Miután ez a fő folyamat lefutott és hiba nélkül működik a következő lépések jönnek:

- A bejelentkezés ellenőrzése
  - felhasználónév és jelszó ellenőrzés
  - maximális próbálkozás ellenőrzés
- Az SQL kód ellenőrzése
  - A kód helyesség ellenőrzése
  - Az adatbázisban a változások ellenőrzése
- A táblák feltöltésének ellenőrzése
  - A kód helyesség ellenőrzése
  - Az adatbázisban a változások ellenőrzése
- Az fa struktúra ellenőrzése
  - A hozzáadás funkció ellenőrzése
    - A gombok működésének ellenőrzése
  - A levelek ellenőrzése
- Az új elem ellenőrzése
  - Az adatok kiírásának ellenőrzése

Minden a listában található elemhez egy teljes teszt eset van hozzárendelve, ezek a teszt esetek önmagukban is funkcionálnak. Ha a teljes funkció ellenőrzésével végeztünk, akkor a használati eset teszt sikeresnek mondható. Minden felhasználói elő követelményhez készíteni kell egy ilyen tesztet.

5. fejezet

A program bemutatása

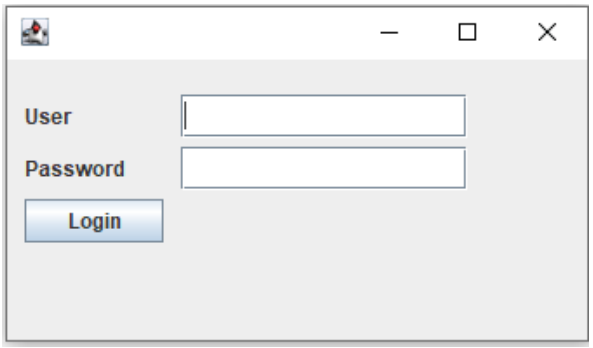
Az előző fejezet után, ebben a fejezetben bemutatom az elkészült programot. Fontos leszögezni, hogy a szakdolgozat célja a tesztelésnek a bemutatása volt, ezért a program is úgy készült, hogy ez legyen szem előtt tartva. Nem véletlenül a legtöbb eddigi és ez utáni teszt eset is a programból lett idézve.

5.1. A program célja

A program egy adott adatbázist kezel. Ez alatt azt kell érteni, hogy a hozzá csatolt adatbázison lehet módosításokat végezni és lekérdezéseket futtatni, illetve egy fa struktúrában szemléltetni lehessen a táblákat.

5.2. Bejelentkezés

A bejelentkezés a program elindítása után az első rész amivel találkozunk. Két mezőt láthatunk amibe belehet írni, illetve angolul felhasználónév és a jelszó szavakat láthatjuk a mezők előtt. Ezen felül egy gomb látható még előttünk, amire angolul a bejelentkezés szó van írva. Továbbá látható egy kis kihagyott rész a panel alján, ahol ha a bejelentkezés nem sikerül a hibaüzenetet láthatjuk.
A bejelentkezési képernyőt az 5.1-es képen láthatjuk:



5.1. ábra. A bejelentkezés

A bejelentkezésre kattintást kezelő kód a következő:

```
@Override
public void actionPerformed(ActionEvent e) {

        String user = loginusertext.getText();
        @SuppressWarnings("deprecation")
        String password = loginpasswordtext.getText();

        if ((user.equals("test") || user.equals("admin"))
                && (password.equals("admin") ||
                password.equals("test123"))) {
                LoginSuccessLabel.setText("Login successful");
                loginFrame.setVisible(false);
                Successgui.successgui();
        } else if (user.equals("test") ||
                user.equals("admin")) {
                LoginSuccessLabel.setText
                ("Login failed. Wrong password");
        } else if (password.equals("test123") ||
                 password.equals("admin")) {
                LoginSuccessLabel.setText
                ("Login failed. Wrong User");
        } else
                LoginSuccessLabel.setText("Login failed");
                tries = tries + 1;
        if (tries == 3) {
                System.exit(0);
        }
}
```

5.3. A kezdőképernyő

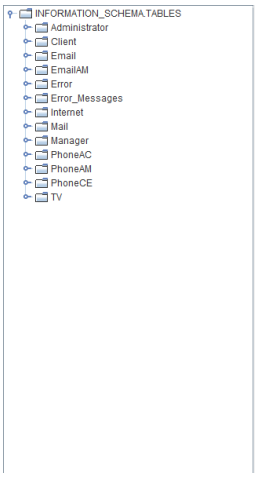
A bejelentkezés után a program kezdőképernyőjét az 5.2-es ábra mutatja. Ezen az ábrán lehet látni, hogy már egy lefuttatott parancs és annak az eredménye megjelent a konzolon. A program használata közben legtöbbször ezt a képernyőt látjuk.

5.2. ábra. A kezdőképernyő

5.4. Fa struktúra

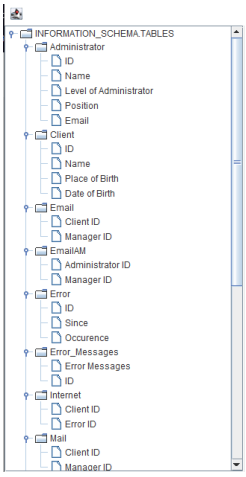
A fa struktúra reprezentálja a csatolt adatbázis felépítését, annak minden táblájával. A tábláknak mappa ikonja van a táblákon belüli oszlopoknak pedig fájl ikonja. Az táblákat a struktúra csomópontként az oszlopokat pedig levélként kezeli, ezért a továbbiakban én is így fogok hivatkozni rájuk.

A fa struktúra alap állapotban az 5.3-mas ábrán látható. Minden csomópontra és levélre rá tudunk kattintani a struktúrában, ami azt eredményezi, hogy a hozzárendelt adatokat kiírja a konzolon és hozzáadja a kiíratás dátumát. Miután egy csomópontra duplán kattintunk, az alatta lévő levelek megjelennek.



5.3. ábra. A fa struktúra

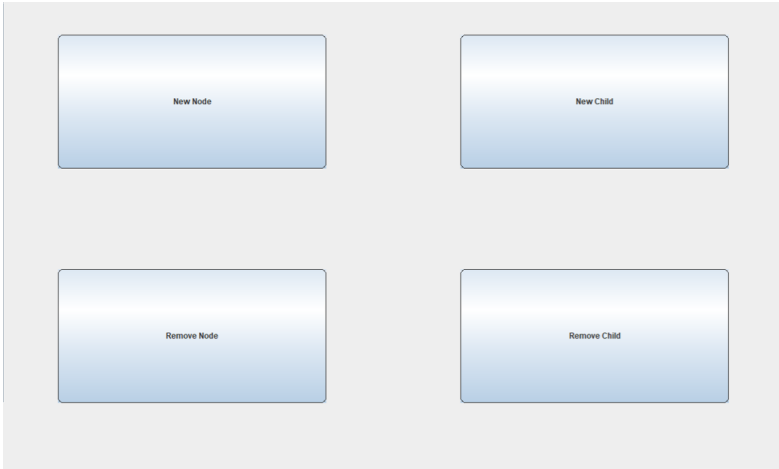
Ha lenyitottuk az összes csomópontot, akkor a túl sok adat miatt, a struktúra jobb szélén egy legörgethető panel jelenik meg, amivel egyszerűen navigálhatunk a csomópontok és levelek között. Erre példát az 5.4-es ábra mutat.



5.4. ábra. A fa struktúra lenyitva

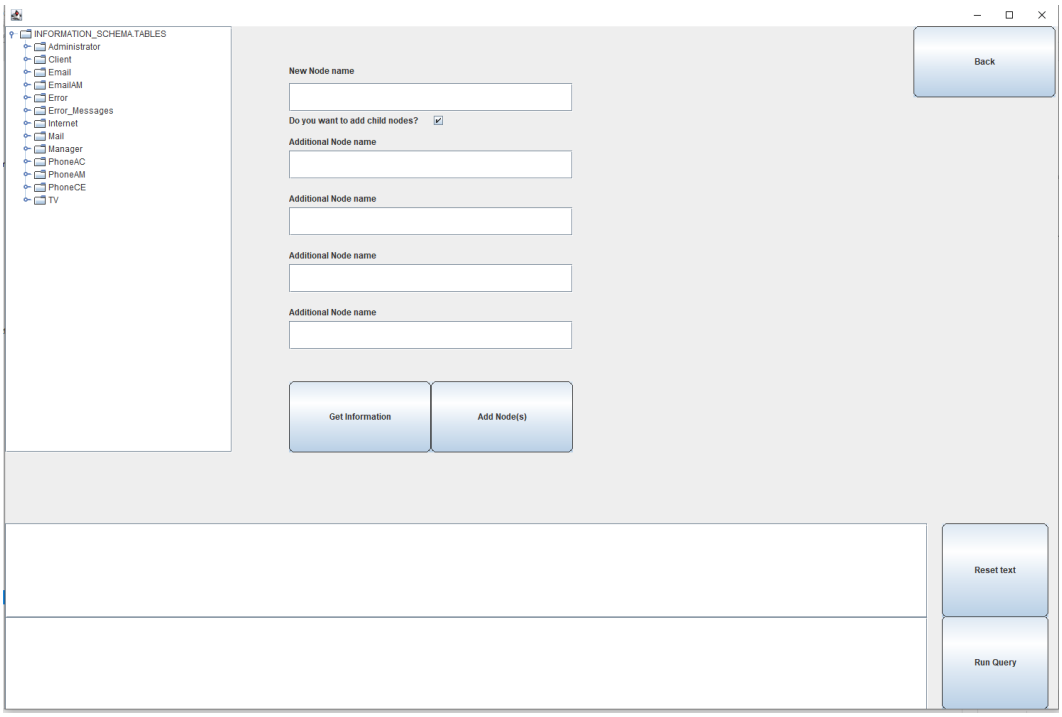
5.5. Fa kezelő gombok

4 gomb látható a kezdőképernyő felső részén, amik a fa struktúra módosítására készültek. Mind a négy gombnak különböző funkciója van ezekre külön-külön kitérek. Ezek a gombok láthatóak az 5.2-es ábrán és ránagyítva az 5.5 ábrán.



5.5. ábra. A kezelő gombok

Ha az új csomópont vagy az új levél gombra kattintunk, akkor eltűnik az alap 4 gomb és az 5.6-os ábrán látható gombok és sorok jelennek meg.



5.6. ábra. Az kezelő felület

5.5.1. Új csomópont

Ezzel a gombbal lehetőségünk nyílik hozzáadni a fa struktúrához egy új csomópontot és ha szeretnénk alá több levelet is. A csomópont hozzáadásához először az információszerzés gombra kell kattintanunk, majd az új csomópont(ok) gombra.
Ha szeretnénk leveleket is hozzáadni, akkor először be kell pipálnunk a jelölőnégyzetet, amivel megjelenik a többi mező, ahova a beírt adatokat már levélként adjuk hozzá. Ehhez ugyanazt kell megtennünk, mintha új csomópontot raknánk hozzá, annyi különbséggel, hogy megjelenik egy új gomb az új csomópont(ok) mellett új levelekként és

arra kell kattintanunk.

5.5.2. Új levél

A gomb működése hasonló az új csomópontéhoz. Először csak 1 mező lesz elérhető számunkra, viszont amint a jelölőnégyzetet bepipáljuk, megjelennek a további mezők. Ahhoz, hogy ez a funkció működjön, a fa struktúrában ki kell jelölni, hogy melyik csomóponthoz vagy levélhez szeretnénk további leveleket hozzáadni. Ha valamelyik mező üresen marad, akkor a program ezt jelzi nekünk a konzolon. Ahhoz, hogy a hozzáadás megtörténjen először az információszerzés gombra kell kattintani, majd az új levelek gombra.

5.5.3. Csomópont törlés

A törlés gombok máshogy működnek mint a hozzáadás gombok. Ha elemet szeretnénk törölni, akkor figyelni kell rá, hogy ez az összes hozzá tartozó levelet is törölni fogja kérdezés nélkül. A törlés menete egyszerű. Ki kell jelölni a fán, hogy melyik csomópontot szeretnénk törölni, majd rá kattintani a csomópont törlése gombra.

5.5.4. Levél törlés

A levél törlés gomb hasonlóan működik, mint a csomópont törlés. A különbség a kettő között, hogy ezzel csak és kimondottan leveleket lehet törölni. Ha egy csomópontot próbálnánk vele törölni, akkor a konzolon megjelenik a hozzá tartozó hibaüzenet. A törlés menete ugyanaz, mint a csomópontnál. Először kijelölésre kerül a levél, majd a gombra kattintás után az eltűnik a fából.

5.6. Konzol

A konzol tartalmaz 2 beviteli mezőt és 2 gombot. A felső beviteli mező csak arra szolgál, hogy a kiadott parancsok eredményét, illetve a hibaüzeneteket kiírja egy időponttal ellátva. A két gomb az alsó beviteli mezővel foglalkozik. A szöveg alaphelyzetbe állítás gombbal az alsó mezőben lévő szöveget ki lehet törölni és vissza állítani alap helyzetbe, amíg a parancs futtatás gombbal ebbe a mezőbe beírt SQL parancsot lehet lefuttatni.

6. fejezet

Automata egység tesztelés

Az automata tesztelésről sok információ nem szerepel a szakdolgozatomban, viszont kihagyhatatlan szerepe van a unit tesztelésnek a szoftverfejlesztési ciklusban. Ezeket a teszteket általában a fejlesztők írják önmaguknak és arra a célra készülnek, hogy a lefejlesztett legkisebb egységeket tesztelje automatikusan. Általában ezzel önmagukat is szokták ellenőrizni, ugyanis ezek úgy készülnek, hogy ha a kód módosításra kerül, akkor is a helyes működést teszteljék. Ha valamelyik unit teszt nem fut le, akkor tudja a fejlesztő, hogy valamit elrontott.

Az én programomhoz is készültek automata unit tesztek. A következő részben ezeket fogom bemutatni. A bemutatásnál találkozunk egy eddig ismeretlen szóval. Ez a JDBC, ami a Java Database Connectivity, ami magyarul a Java adatbázis-csatlakoztatás-ként fordítható. Ez felel a program és az adatbázis közötti kommunikációért.

6.1. Bejelentkezés

A bejelentkezés egység tesztjét két formában is bemutatom. Az első rész az a sikeres teszt, a második rész pedig egy sikertelen teszt. A sikertelent úgy kell érteni, hogy szándékosan negatív kimenetelt várunk el. A teszt a két beviteli mezőbe beírandó felhasználó név és jelszó kombinációt megkapja és rákattint a bejelentkezés gombra. Miután a gombra rákattintunk, a program vagy beenged minket, vagy hibaüzenetet kapunk. Ez alapján eldönti az egység teszt, hogy sikeres-e vagy sem. Az első bejelentkezés futási ideje 0,163 másodperc, amíg a másodiké 0,144 másodperc.

```
@Test
final void testActionPerformed() {
        gui.logingui();
        loginusertext.setText("test");
        loginpasswordtext.setText("test123");
        loginButton.doClick();
        assertEquals("Login successful",
        LoginSuccessLabel.getText());
}

@Test
final void testActionPerformed_fail() {
```

```
        gui.logingui();
        loginusertext.setText("test");
        loginpasswordtext.setText("test1234");
        loginButton.doClick();
        assertEquals("Login failed. Wrong password",
        LoginSuccessLabel.getText());
}
```

6.2. Új csomópont hozzáadása

Ezzel az egység teszttel az új csomópont funkciót tesztelem. Létrehozásra kerül egy új fa struktúra az egység tesztben, és ehhez a fához ad hozzá egy új csomópontot az egység teszt. Ameddig ez a teszt sikeresen lefut, addig a funkcióval nem lesz gond a futtatás alatt. A teszt futási ideje 0,002 másodperc.

```
@Test
final void testNewParentNode() {
        DefaultMutableTreeNode Administrator =
         new DefaultMutableTreeNode("Administrator");
        DefaultMutableTreeNode Client =
         new DefaultMutableTreeNode("Client");
        DefaultMutableTreeNode Email =
         new DefaultMutableTreeNode("Email");

        Administrator.add(Email);
        Administrator.add(Client);
        String name = "new";

        Jtree.tree = new JTree(Administrator);
        Jtree.newParentNode(name);
        assertEquals(3,Administrator.getChildCount());
}
```

6.3. JDBC csatlakozás

A csatlakozás egység tesztje viszonylag egyszerűnek tűnik, de a kód mögött futó rész eléggé kacifántos. A megadott adatok alapján meghívja a tesztelendő funkciót, ami először összekapcsolódik az adatbázissal, majd lefuttatja a parancsot és annak az eredményét eltárolja és visszaadja, majd pedig kiírja a konzolba. Ezt a visszakapott adatot ellenőrzi le a teszt és ha megfelel az elvártnak akkor sikeres. A teszt futási ideje 0,322 másodperc

```
@Test
final void testJdbc() {

        String sql = "Select * from Client";
        String column = "ID";
```

```
        logArea = new JTextArea("");

        Jtree.jdbc(sql, column);

        assertEquals("1, 2, 3, 4, 5, "
         + formatter.format(date) + "\n",logArea.getText());

}
```

6.4. JDBC tábla készítés

A tábla készítés egység tesztje hasonlít a csatlakozáséra kinézetben, de a mögötte lévő kód változik. Itt is előre megadásra kerülnek az adatok, és meghívódik a tesztelendő funkció, viszont itt az eltárolt információ teljesen más, mint az előzőnél. Az itt visszajövő érték nem egész szám, hanem egy szöveg, így az kerül kiírásra, majd pedig összehasonlításra az elvárt eredménnyel. A teszt futási ideje 0,032 másodperc

```
@Test
@Order(5)
final void testJdbcNodeCounterCreate() {
        String sql = "Create table B (id int, id2 int);";
        String command = "Create";

        logArea = new JTextArea();


        Jdbc.JdbcNodeCounter(sql, command);

        assertEquals("The table was created"+
         formatter.format(date) + "\n",logArea.getText());

}
```

7. fejezet

Összefoglalás

7.1. Értékelés

A dolgozat célja a szoftver tesztelés bemutatása volt. Ez alatt értendő, hogy milyen tesztelési módszereket mikor érdemes használni és, hogy mik ezek a módszerek.

A szakdolgozatban számos módszerrel találkozhattunk, amiket szöveges leírás után gyakorlati példával is bemutattam, így egy átfogóbb képet kaptunk a módszerekről. Készült egy adatbázist kezelő program a szakdolgozat mellé, aminek a célja a módszerek alkalmazása a gyakorlatban volt, így számos funkció ennek megfelelően készült el, ami lehetséges, hogy egy cégnek készült szoftverben teljesen máshogy lenne implementálva.

A tervezettnél a programban az adatbázissal összeköttetéssel foglalkozó funkciók jobban készültek el. Sikerült egy funkciót megírni, ami le tudja kezelni önmagában a különböző parancsokat, ahelyett, hogy parancsonként külön-külön funkcióra legyen szükség.

A tervezettnél rosszabbul a fa struktúra elkészítése sikerült. Szerettem volna, hogy adatbázis független legyen a program és bármilyen hozzácsatolt adatbázissal működjön, viszont végül a tesztelés szempontjából, jobbnak láttam, hogyha egy hozzá elkészített adatbázissal foglalkozik csak a program.

7.2. További tervek

Mindenképpen szeretném tovább fejleszteni a programot. A tervek közé tartozna, hogy lehessen vele adatbázist kreálni és törölni, így teljesen adatbázis függetlenné tenni a programot, illetve több adatbázissal is szeretném, ha tudna egyszerre foglalkozni.

Ezen felül szeretnék a grafikus felületen szépíteni, illetve még plusz elemeket hozzáadni, így a jelenlegi 4 gombos rész helyett, egy teljes panelt hozzáadni a jobb széléhez, ahol ezek a gombok elérhetőek lennének és a felszabadult helyen, pedig az éppen megnyitott adatbázis elemeit lehetne látni.

Irodalomjegyzék

[1] Meile Posthuma Klaus Olsen and Stephanie Ulrich. Official istqb® ctfl syllabus - magyar, 2018.

[2] Molnár Richárd. Mit csinál a tesztelő?

[3] Thomas Hamilton. What is adhoc testing?

[4] Meile Posthuma Klaus Olsen and Stephanie Ulrich. Official istqb® ctfl syllabus - magyar, 2018.

[5] Meile Posthuma Klaus Olsen and Stephanie Ulrich. Official istqb® ctfl syllabus - magyar, 2018.

[6] Meile Posthuma Klaus Olsen and Stephanie Ulrich. Official istqb® ctfl syllabus - magyar, 2018.

CD Használati útmutató

Ennek a címe lehet például *A mellékelt CD tartalma* vagy *Adathordozó használati útmutató* is.

Ez jellemzően csak egy fél-egy oldalas leírás. Arra szolgál, hogy ha valaki kézhez kapja a szakdolgozathoz tartozó CD-t, akkor tudja, hogy mi hol van rajta. Jellemzően elég csak felsorolni, hogy milyen jegyzékek vannak, és azokban mi található. Az elkészített programok telepítéséhez, futtatásához tartozó instrukciók kerülhetnek ide.

A CD lemezre mindenképpen rá kell tenni

- a dolgozatot egy `dolgozat.pdf` fájl formájában,
- a LaTeX forráskódját a dolgozatnak,
- az elkészített programot, fontosabb futási eredményeket (például ha kép a kimenet),
- egy útmutatót a CD használatához (ami lehet ez a fejezet külön PDF-be vagy MarkDown fájlként kimentve).