INNOVÁCIÓ A SZOFTVERTESZTELÉSBEN – MEGOLDÁS-E AZ AUTOMATIZÁLÁS?

Béli Marcell – Tóth Fanni – Varga Tamás

Absztrakt: A tanulmány célja az automatizált szoftvertesztelés elméleti és gyakorlati hatékonyságának vizsgálata. Ehhez felmérjük, milyen szempontok szerint érdemes automatizált tesztelői szoftvert választani, valamint rávilágítunk, milyen esetekben hatékonyabb az automatizált tesztelés a manuális teszteléssel szemben. Előbb elméleti síkon vetjük össze a két tesztelési módszertant, majd egy valós üzleti esettanulmányon keresztül mutatjuk be a tesztelői szoftver kiválasztását és végzünk el egy gyakorlati összehasonlító elemzést. Öt automata tesztrendszer összehasonlítása alapján kiderül, hogy az alapvető technológiai és gazdasági követelményeken felül a rugalmasság és a megbízhatóság is kiemelkedő szempont a kiválasztási folyamatban. A tanulmányból kiderül az is, hogy az automatizálás csak ismétlődő tesztelések esetén lehet hatékony. Egy legalább tíz ügyféllel rendelkező szoftvergyártónak azonban megéri az automatizált tesztelés irányába mozdulni.

Abstract: The aim of our study is to investigate the theoretical and practical effectiveness of automated software testing. We assess the criteria for choosing automated testing software and highlight the cases in which automated testing is more effective than manual testing. First, we compare the two testing methodologies on a theoretical level, then we present a real business case study to illustrate the selection of a testing software and perform a practical comparative analysis. A comparison of five automated test systems reveals that, in addition to basic technological and economic requirements, flexibility and reliability are also key considerations in the selection process. The study also shows that automation can only be effective in the case of repetition. However, for a software developer with at least ten customers, it is worthwhile to move towards automated testing.

Kulcsszavak: szoftvertesztelés, automatizálás, innováció, esettanulmány

Keywords: software testing, automation, innovation, case study

1. Bevezetés

A szoftver tesztelése elengedhetetlen része a szoftverfejlesztésnek. A fejlesztők ugyan tökéletes rendszerek kiépítésére törekednek, de bármikor kerülhet hiba a kódba. A tesztelési fázis legfontosabb funkciója a kritikus hibák kiszűrése, amelyek nem kerülhetnek a végfelhasználók elé. A manuális tesztelési módszer mellett innovatív automatikus tesztelési technika is bevezethető a tesztelési folyamatba. A tanulmányban három kérdésre kívánunk választ nyújtani: 1) mely szempontok szerint célszerű automatizált tesztelői programot választani; 2) a manuális vagy az automatizált szoftvertesztelés a hatékonyabb tesztelési forma; 3) ebben a jelentősen felgyorsult világban milyen esetekben van szükség manuális tesztelésre. Ezekre a kérdésekre összehasonlító módszertan, valamint egy valós üzleti esettanulmány segítségével válaszolunk. Az FX Software Zrt. által fejlesztett InFoRex több mint 20 éves múlttal rendelkező, hazai fejlesztésű, banki és vállalati integrált treasury rendszer. A szoftver maga egyrészről kellően komplex, másrészről kellően fontos pozíciót tölt be a hazai és regionális piacon, hogy egy valós üzleti problémát járhassunk körül és a piaci szereplők számára releváns megállapításokat tegyünk.

ISSN 1788-7593 (Nyomtatott) DOI: https://doi.org/10.14232/jtgf.2022.3-4.47-66
ISSN 2676-9867 (Online)

Az automata tesztrendszer kiválasztásához összehasonlító elemzést végeztünk. A felállított előzetes követelményrendszerünk alapján, mint például az eredmények riportálhatósága vagy a tanulhatóság, öt programot hasonlítottunk össze. A választás a TestComplete programra esett, mivel az alapvető technológiai és gazdasági követelményeken felül további előnyös tulajdonságokkal is bír, mint a megbízhatóság vagy a rugalmasság.

A manuális és automata teszt összehasonlítása az InFoRex rendszer egy tesztesetének tesztelésén keresztül valósul meg. Kiderül, hogy az automata tesztet minden esetben megelőzi egy manuális tesztelés. Az automatizált tesztelés esetén csak akkor érhetünk el hatékonyságot, ha a tesztelés ismételhető. A bemutatott példán is látszik, hogy egyszeri alkalom esetén a manuális tesztelés gyorsabban elvégezhető. Ismétlődő tesztelés esetén azonban már pár ismétlés után az automatizált tesztelés a hatékonyabb megoldás. Többször futtatható, stabilabb tesztek készítéséhez több idő szükséges, de egy legalább tíz ügyféllel rendelkező szoftvergyártónak (mint például az FX Software) már egyetlen átfogó tesztelési kör alatt is megéri az automatizált tesztelés irányába mozdulni.

A tanulmányban a kapcsolódó szakirodalom bemutatását követően összevetjük a manuális és automatizált tesztelés elméletét a gyakorlattal. Ezt követően egy esettanulmányon keresztül bemutatjuk, milyen szempontok szerint érdemes tesztelő programot választani és kitérünk az összehasonlításba vont szoftverek részletes bemutatására. Gyakorlati példán vezetjük le egy teszteset manuális és automatizált tesztelésének folyamatát, majd kiértékeljük a két tesztelési forma teszteredményét. Végezetül pedig összefoglaljuk eredményeinket és levonjuk következtetéseink.

2. Szoftvertesztelés a szakirodalomban

A Nemzetközi Szoftvertesztelési Képesítési Testület (ISTQB) hivatalos meghatározása szerint a szoftvertesztelés: a szoftvertermékek és a kapcsolódó munkatermékek tervezésével, előkészítésével és értékelésével kapcsolatos valamennyi, statikus és dinamikus életciklus-tevékenységből álló folyamat, melynek célja 1) annak megállapítása, hogy szoftvertermékek megfelelnek-e a meghatározott követelményeknek; 2) annak bizonyítására, hogy alkalmasak a célra, amire készültek, és 3) a hibák észlelése (ISTQB, 2021).

Másképpen megfogalmazva a szoftvertesztelés egy olyan folyamat vagy folyamatok sorozata, amelynek célja, hogy megbizonyosodjunk arról, hogy a számítógépes kód azt teszi, amire tervezték, és hogy nem tesz semmi nem szándékolt dolgot. A szoftvernek kiszámíthatónak és következetesnek kell lennie, nem okozhat meglepetéseket a felhasználóknak (Myers et al., 2012).

A nem megfelelő és nem hatékony tesztelés felelős a szoftverek megbízhatóságával kapcsolatos számos problémáért, amellyel a felhasználók szembesülnek. Mivel a szoftverek egyre összetettebbek, a teljes és kimerítő tesztelés egyre nehezebbé vált. Korábban a szoftver tesztelése csupán másodlagos feladatot jelentett, de napjainkban a tesztelés számos alkalmazási területen a termékfejlesztés kiindulópontját jelenti, költsége a termékfejlesztés költségeivel vetekszik (Catelani et al., 2011). Becslések szerint a forráskód hibáinak felderítésére alkalmas

szoftvertesztelés a fejlesztés több mint 50%-át teszi ki, de a tesztelésre fordított idő és költségek az automatizált teszteléssel csökkenthetők (Diaz et al., 2003).

A szoftvertesztelés sarokköve a teszt folyamatának megtervezése. Első lépésben a stratégiát alapozzuk meg. Ez megnyilvánul az erőforrások meghatározásában, a környezeti tényezőkhöz való alkalmazkodásban, illetve a célok és irányelvek kitűzésében. Második lépésként a teszt környezetének analizálása és kialakítása következik, ami még szintén a tervezési fázishoz köthető, de itt már konkrét lépéseket határozunk meg, például: mit kell tesztelni, tesztesetek megtervezése, tesztkód írásának megtervezése. Harmadik lépésben a megvalósításra és a végrehajtásra térünk át, melynek során meghatározzuk a teszt fajtáját. Ebben a szakaszban döntés születik arról, hogy az adott teszteset tesztelésének megvalósítása manuális és vagy automatikus formában hatékonyabb-e. Futtatjuk a tesztet, majd naplózzuk a következő adatokat: idő, verzió, adatbázis, eredmény, hibák. Végezetül riport készítésével és a teszt lezárásával döntést hozunk az eredményről, levonjuk a konklúziókat és mérlegeljük a további szükséges lépeseket, javításokat.

A tesztelés különböző szinteken valósul meg. Unit vagy Modul teszt a legalapvetőbb tesztelési forma. Ezen a szinten, a szoftvert felépítő egységek vizsgálata történik meg és ezzel ellenőrizhető, hogy a kód építőkövei megfelelően működnek-e. Azt ellenőrizzük, hogy az eltérő bemenetek esetén a megfelelő kimenetet kapjuk-e meg. Következő szint az integrációs teszt: ezen a szinten a program fejlesztése közben felhasznált (új, meglévő vagy harmadik fél által készített) komponensek együttműködését teszteljük. Következő szint a rendszer teszt, ahol a funkcionális és nem funkcionális elvárásoknak való megfelelést teszteljük. Ide sorolhatóak például a smoke teszt, performancia teszt, biztonsági tesztek, használhatósági tesztek. Végezetül megkülönböztetünk elfogadási teszteket, melyek lehetnek belső vagy külső (ügyfeles) tesztek, irányulhatnak egy funkcióra vagy akár a rendszer egészére (UAT / Béta tesztelés). A fázis célja a már kész rendszer elfogadása, valós vagy végleges környezetben teszteljük, valós adatokkal (STF, 2020).

A tesztelés négy főbb modellje különböztethető meg (Tompa, 2019). A vízesés modell szerinti tesztelés során lépcsőzetesen kapcsolódnak egymáshoz az egyes tesztelési fázisok. Az első lépés a követelmények meghatározása és elemzése, melyet a rendszer architektúrájához illeszkedő tesztelési lépések megalkotása követ, összhangban a korábban felsorolt tesztelési szintekkel. A V-modell tesztelés, melynek a „v” betűhöz hasonló az alakja, módosított vízesés modellnek tekinthető. A tesztesetek kitalálása nem a folyamat végén, hanem a specifikációval egyszerre történik. Amint elkészül egy mérföldkő, a fejlesztési oldalon vele párhuzamosan a tesztelése is elkészül (például a build-elhetőség előfeltétele a megfelelő unit teszt lefedettség). Ez a modell használatos, ha a termék moduláris felépítésű, párhuzamosíthatóak a tesztesetek: miközben az egyik modulban folynak a unit vagy komponens tesztek, a másik modulban átvételi teszt is végezhető. A harmadik főbb modell az iteratív-inkrementális módszertanok, melyek lényege a folyamatos, kis lépésekre bontott követelmény meghatározás, tervezés, implementálás és tesztelés. Egy előzetes terv szerint, fokozatosan épül fel a rendszer és a tesztelés folyamata.

Ide sorolható a prototípus modell vagy a gyors alkalmazásfejlesztés (RAD). Végezetül, az agilis modell tesztelés a jó módszer akkor, ha változik a rendszer és nem kötött. A fejlesztéssel párhuzamosan halad a tesztelés is és alkalmazkodik az ügyfelek változó igényeihez. Ide sorolhatjuk a Scrum vagy az extrém programozás (XP) módszertanokat.

2.1. Manuális tesztelés elméletben és gyakorlatban

Mint a nevéből is következik, ennek a módszernek az alkalmazása során kézzel szükséges végrehajtani a tesztelést és az eredményről készülő riport is manuálisan jön létre.

A manuális tesztelés a legalapvetőbb szoftvertesztelési technika, ezzel a módszerrel detektálhatók az apróbb, rejtett hibák is, amelyek az automatizált tesztelés során nem feltétlenül merülnek fel. Egy újonnan épült rendszert mindenképpen először kézileg kell letesztelni, amely gyakran sok erőforrást igénylő feladat, de biztosítja a szoftver későbbi optimális működését és előkészíti, hogy a fejlesztési folyamat későbbi szakaszaiban automatizált teszteket is lehessen futtatni (Javatpoint, 2021). A legtöbb automatizált tesztet először manuálisan kell végrehajtani, hogy megismerjük a rendszer működését. Ezeknek az információknak a birtokában lehet megírni a kódot a tesztesetek automatizált létrehozására. Az információs technológia jelenlegi fejlettségi szintjén száz százalékosan automatizált tesztet nehéz létrehozni, mert a lehetséges interakciók számossága miatt a teljes rendszert nem lehet lefedni vele. Egy következő nagy innovációs ugrás lehet az automata tesztelések esetén az AI technikák alkalmazása, amellyel száz százalékos lefedettség nélkül, a mesterséges intelligenciára támaszkodva, előre fel nem mért esetekből eredő hibák is megtalálhatóak lehetnek gépi tesztelés során. Jelenleg ilyen megoldás még nem áll rendelkezésre a piacon.

A következőkben három manuális tesztelési technikát mutatunk be, részletezve ezek működési elvét, elsődleges célját, előnyeit és hátrányait, valamint fő alkalmazási területüket Javatpoint (2021) alapján. A fehér doboz technika esetében szerkezeti tesztelésről beszélünk, melynek során a belső kódokat, infrastruktúrát teszteli maga a fejlesztő. Elsődleges eszközei a unit tesztek. A fekete doboz módszernek a használata során funkcionálisan teszteljük a szoftvert, tehát a kód nem látható, csak a specifikáció. Ügyfélként teszteljük a rendszert, egy bemeneti érték alapján megvizsgáljuk, hogy a várt eredményt kapjuk-e. A fekete és fehér doboz tesztelésnek a kombinációja a szürke doboz technika, amely a két módszer ötvözete. Ennek alkalmazása során hozzáférünk a belső kódoláshoz, a gyakorlati tesztelés pedig a funkcionalitás szintjén zajlik. A szürke doboz tesztelés előnyei közé tartozik többek között, hogy a teszteseteket nem szükséges a forráskódból tervezni, létrehozhatóak a fekete doboz módszertan szerint is. A szürke doboz technika esetén a bemeneti és kimeneti értékek mellett ellenőrizzük az eredményeket és kiemelt utakat, amelyeken végig halad a teszt. Szükség esetén bővítjük a bemeneti értékkészletet, hogy a kód minden részletének tesztelése megvalósuljon.

A háromfajta technika ismertetése után a következőkben a manuális tesztelés gyakorlati lépéseinek bemutatása következik az InFoRex szoftver esetén: először a

rendszer dokumentációját ellenőrizzük, ez alapján kiválasztjuk a tesztelni kívánt területeket (vagy konkrét funkció tesztje esetén fordítva). Ezután elemezzük azokat a követelményeket, amelyeket a felhasználó elvárhat a rendszer beüzemelésénél (milyen funkciókra van licence, milyen interface-ek működése szükséges). Következő lépésben létrehozzuk a teszteseteket a követelménydokumentáció alapján, majd az említett technikákkal (függően attól, hogy a fejlesztés melyik lépésében kapcsolódik be a tesztelés) elvégezzük a tesztelést. A felmerülő hibákról feljegyzést készítünk és továbbítjuk a fejlesztőknek. Végül, ha a hibák javítása megtörtént, újra teszteljük a rendszert az egész folyamatot ismételve.

2.2. Automatizált tesztelés elméletben és gyakorlatban

Az automatizált tesztelés ismétlődő tesztelések esetén jelentős költséghatékonyságot jelenthet. Egy hosszabb, nehezebb, sok figyelmet igénylő, aprólékos tesztet megírva, a gép már könnyedén feldolgozza a folyamatot, ezzel időt és energiát spórol meg a tesztelőknek és ezzel együtt a vállalatnak.

De mi is pontosan a tesztautomatizálás? Tesztesetek automatikus futtatásának sorozata, melynek célja a szoftverminőség javítása a tesztadatok és teszteredmények felhasználásával. (Testim, 2019) A folyamat minőségbiztosítást végez, de csak akkor lehet igazán jövedelmező, ha a környezeti (szerver és összetevői) és működési feltételeket helyesen mérlegeljük, és időtálló teszteseteket készítünk. Ellenkező esetben a ráfordított erőforrások nem térülnek meg, sőt, veszteség is származhat az automatikus tesztek fejlesztéséből.

Az automatizált tesztelés módszerétől elvárható, hogy a teszt többször ismételhető legyen, hiszen aránytalanul nagy erőforrás felhasználást igényel egy olyan teszt megírása, amit csak néhány alkalommal lehet futtatni. A teszt ismételhetőségének legfontosabb feltételei közé tartozik a teszt adatainak és környezetének beállítása. A környezet stabilizálása érdekében a teszthez hozzá kell adni a megfelelő felhasználókat és azok jogosultságait. Ezután végezzük el a tesztet, majd elemezzük az eredményt. Végül az adatokat és a környezetet visszaállítjuk az eredeti, tehát a teszt indítása előtti állapotukba a későbbi futtatásokhoz. Az automatizálás következő fontos jellemzője a determinisztikus jelleg. Leegyszerűsítve ez azt jelenti, hogy a tesztek eredményeinek egyeznie kell egymással azonos bemenetek esetén. A szoftvereknél ezt nehéz elérni, mivel végtelen számú bemenettel nehéz ugyanazt az eredmény reprodukálni. Kiemelendő, hogy emberi véleményt nem lehet automatizálni, kódolni (Testim, 2019).

A következőkben arra térünk ki, hogy melyek azok a fajta tesztek, amelyeket érdemes főként automatizáltan végezni. A unit tesztek build szerveren (ahol a forráskódból a kiadható verzió előállítása történik) a kód változásakor automatikusan futnak. Az integrációs tesztelési metódusnál lényeges, hogy a teszt kimenetele ne függjön külső tényezőktől. Amennyiben a fejlesztési folyamat része az újonnan készült funkciók automata teszttel történő lefedése, akkor ide sorolhatjuk az átvételi (Acceptance) tesztet is. Az itt elkészült tesztelő kód később a regressziós teszt része lehet. Az automatikus regressziós tesztet folyamatosan érdemes futtatni, időszakosan, a szoftver naprakészen tartása érdekében és a hibajavítások, valamint a

fejlesztések okozta esetleges problémák kiszűrése miatt. A performancia tesztek futhatnak a regressziós tesztek részeként, amely magában foglalja a teljes funkcionális és integrációs tesztet, vagy külön teszteset csomagból is összeállhatnak. Fontos megemlíteni, hogy ilyen típusú tesztelés esetén a megfelelő – valós működési környezetet imitáló – környezet megléte kritikus pont. Végezetül külön kiemelnénk a smoke tesztet, amely egy jól leválasztható része a teljes regressziós tesztelésnek. Célja, hogy ellenőrizze, hogy minden szerviz és függőség megfelelő-e, a felületek megnyílnak, frissülnek, de mélyebb funkcionális teszt nem történik. A regressziós tesztek részeként időszakosan is fut, illetve verzió kiadása előtt is mindenképpen érdemes futtatni. (Testim, 2019)

Hogyan is zajlik a gyakorlatban tömören az automatizált tesztelés folyamata? A gyakorlati folyamat megegyezik Testim (2019) által megfogalmazottakkal: először előkészítjük a környezetet, ahol a teszt egy előre megírt forgatókönyv szerint fog lefutni. A második lépésben a gép futtatja a tesztet önerőből. Végül elemezzük a kapott eredményt. A teszt automatizált jelentést ír, viszont visszamenőleg le tudjuk ellenőrizni az egész tesztet lépésről lépésre.

2.3. Manuális és automatizált tesztelés összehasonlítása

Az *1. táblázat* elméleti szinten tekinti át a két tesztelési típus előnyeit és hátrányait különböző szempontok szerint. Később ennek gyakorlati oldalára is rávilágítunk.

Amikor egy adott tesztelési típus használata mellett döntenénk, fontos figyelembe venni az *1. táblázatban* foglaltakon túl, hogy milyen erőforrással rendelkezünk, képesek vagyunk-e például megfelelő automatizált teszt szkripteket készíteni.

De mit is jelent az, hogy „megfelelő” egy automata teszt szkript? A legtöbb automata tesztelő eszköz már rendelkezik teszt felvétel és visszajátszás funkcióval. Azonban ezek az egyszerűen felvett szkriptek a legtöbb esetben nem tartósak, akár már a rögzítés után sem lehet őket újra futtatni, csak pontosan ugyanazon a verzión és állapoton, amin a rögzítés történt. Mindemellett a kódjuk strukturálatlan, így nehezen karbantartható. Természetesen a rögzített szkriptekből kiindulva, ezek a hibák javíthatók, a szkript paraméterezhetővé, többször meghívhatóvá fejleszthető. Kiegészíthető logikai ellenőrzésekkel, kezelhetővé tehető olyan összetett futtatásokra, amiben több felhasználónak szükséges egymásra épülve interakciókat végeznie vagy például egy másik szkriptre épülve kell működnie egy adott tesztesetnek. De ehhez természetesen nagyobb szkript fejlesztői erőforrás szükséges. Minden cég a saját igényei és képességei alapján döntheti el a manuális teszt, a rögzített automata teszt és a fejlesztett automata teszt skálájának melyik részét tekinti optimálisnak.

1. táblázat: **Manuális és automatizált teszt előnyei és hátrányai**

Szempontok	***Automatizált tesztelés***	***Manuális tesztelés***
Termelékenység	Legnagyobb előnye, hogy mindig ugyanúgy teljesít. A gépnek nincs „rossz napja”: annyi tesztet készít el, amennyit megadtunk. Pontos, nem változtat a sorrenden.	Időigényesebb, hibázási ráta magasabb, hiszen ember végzi.
Konzisztencia	Reprodukálhatóak a tesztek. Pontos és bármennyiszer megismételhető a teszt.	A bemeneti adatokat nem mindig dokumentáltak részletesen. A vizsgálat lépéseit nem 100 százalék-os biztonsággal tartják be. Ennek köszönhetően néha nehéz reprodukálni a meghibásodás helyét és okát. Az eredmények eltérőek lehetnek.
Teszt végrehajtása	Automatizált tesztelő program végzi.	Humán erőforrás végzi.
Gyorsaság, kivitelezés	Olyan gyorsasággal tud futni, amit a környezet enged.	Az emberi képességeken és a tapasztalatokon múlik a gyorsaság. Fontos megemlíteni, hogy nem csak maga a teszt gyorsasága a lényeges, amíg a tesztelő tesztel, addig nem tud új teszteseteket írni, ezekre külön kell időt szánnia.
Lefedettség	Könnyedén lehet átfogó, biztos vizsgálatot végezni.	Nehéz minden apróságra odafigyelni és lefedni a különböző modulokat.
Fenntartás, költségek	Tesztek bármikor futtathatóak, este, napközben vagy akár ünnepnapokon is. Nincs extra költsége. Magát a programot beszerezni, illetve fenntartani egy kezdeti, egyszeri nagyobb összegű beruházást jelent, viszont kifizetődő, ha megtaláljuk a megfelelő módját a használatának.	Drága fenntartani egy tesztelői csapatot, betanítani, megfelelő munkaeszközöket biztosítani, hozzáféréseket biztosítani a szoftverhez. Számos extra költséggel jár, például túlóra díj, ünnepnapok, szabadságok.
Eredmény formaisága	Automatikusan generálódnak. Lényeges, hogy nem csak a hibákat írja le, hanem az egész tesztet. Az eredményt viszont érdemes fenntartásokkal kezelni, hiszen úgy tűnhet, hogy megfelelően működik a program, azonban hibázhat a szoftver is.	A teszteket végig csinálni, elemezni sok időt őröl fel. Előfordul, hogy csak akkor készül róla tesztjegyzőkönyv, ha hibás volt a végkimenetel. Erre mindig kellene figyelni, például viszonyítási alapnak is tökéletes egy jó teszt. A vezetőknek vagy ügyfeleknek is lehet prezentálni, hogy teljesített a rendszer. További veszélye, hogy a tesztelő hibázhat, amikor leírja, hogy pontosan mit is nyomott meg, mielőtt egy hibába ütközött, ami félrevezeti a fejlesztőt. Az eredmény leírását nehéz megszabni, hiszen legtöbb probléma eltérő, sémákkal nehezen lehet lefedni.
Ad Hoc és Exploratory tesztelés	Nem képes ad hoc tesztelésre. Nincs önálló döntése. Javítani, optimalizálni nem tudja a tesztet.	Alkalmas mindkettőre, megvan a szabad akarat és az ítélőképesség.
Dinamikus alkalmazások	Magas szintű karbantartás szükség a szkriptekhez. Ezek lehetnek modulárisak, újra felhasználható szkriptek, függvények.	Rugalmasabban alkalmazkodik a dinamikus alkalmazásokhoz. Nagy szorgalomra és tapasztalatra van szükség ezek elvégzéséhez.

Forrás: Applabs (2008) alapján saját szerkesztés.

3. Esettanulmány

Az elméleti felvezetés után, nézzünk meg egy konkrét példát, melyik szoftvert és hogyan alkalmazza az FX Software Zrt. az InFoRex szoftver teszteléséhez! A manuális tesztelés a kezdetektől fogva jelen van a cég életében, de a rendszer növekvő összetettsége és az ügyfelek magasabb szintű kiszolgálása érdekében szükséges volt az automatizált tesztelés kiépítése. Az InFoRex több mint 20 éves múlttal rendelkező, hazai fejlesztésű banki és vállalati integrált treasury rendszer. A szoftver maga egyrészről kellően komplex, másrészről kellően fontos, hazai piacon piacvezető pozíciót tölt be, valamint regionálisan is jelen van a piacon, hogy egy valós üzleti problémát járhassunk körül és a piaci szereplők számára releváns megállapításokat tegyünk.

3.1. A megfelelő tesztelő program kiválasztása

Az elmúlt években az informatika robbanásszerű fejlődése elérte az automata tesztelő rendszereket is. Így már elvárható, hogy megoldást nyújtsanak a folyamatos integrációra és a DevOps módszerekre, amelyeknek minőségben és gyorsaságban is előrelépést tudnak szolgáltatni. Fontos, hogy megbizonyosodjunk arról, hogy kihasználjuk az automatizálás előnyeit a választott eszközzel, illeszkedjen a cég meglévő rendszereihez, működéséhez. Szintén manapság már elvárható, hogy több olyan kényelmi funkcióval legyenek felvértezve, amelyek a tesztírók napi munkáját nagyban meg tudják könnyíteni.

A szempontok, amelyek alapján a megfelelő automata tesztelői szoftvert kiválasztottuk az InFoRex-hez: 1) WinForms és DevExpress (mint a felület alkotó kontrolok) támogatás alapkövetelmény. 2) Eredmény riportálhatósága (összesítve vagy részleteiben) belső felhasználásra és ügyfelek számára. 3) CI (folyamatos integráció) kompatibilitása a meglévő fejlesztői és verziókövetési rendszerekhez. 4) Tanulhatóság: szenior fejlesztői vagy mélyszakértői tudás szükséges-e a használatához. 5) Licenc és a fenntartás költségei.

Az automata tesztrendszer kiválasztásához összehasonlító elemzést végeztünk. Első lépés természetesen egy online keresés és előszűrés a megfelelő rendszerek listájáról. Az elsődleges szűrés eredményeképp öt programot hasonlítunk össze: Visual Studio Coded UI Test Builder, Appium, Ranorex, TestArchitect és a TestComplete.

A Visual Studio Coded UI Test Builder Windows-alapú tesztelő eszköz. Webes és asztali funkciók tesztelését, valamint teljesítménytesztelést egyaránt végez. Lehetőséget nyújt kódalapú tesztelésre, kódhasználat nélkül. Előnyei, hogy Visual Studioba épített, melyet az FX Software is használ, továbbá könnyen szerkeszthető és kódolható, valamint a kódok könnyen másolhatók és adaptálhatók. Hátrányai a tesztelői eszköznek, hogy fejlesztői tudás és környezet szükséges, ami extra erősforrásokat igényel, valamint nehezen integrálható CI folyamatba. (**Microsoft**, 2021)

Az Appium leginkább mobilwebhez alkalmas, illetve hibrid alkalmazásokhoz IOS/Androidhoz kompatibilis, de természetesen asztali platformhoz is használható. Lényeges előnye, hogy nyílt forráskódú, bárki hozzáférhet, könnyű konfigurálni,

vagyis felhasználóbarát és sok programnyelvet támogat, ezek a Java, Python, Ruby, C#. Hátrányai, hogy fejlesztői tudás szükséges, és nem garantált a DevExpress későbbi kompatibilitása, ami esetünkben elengedhetetlen, továbbá lassan fut a rendszer és a dokumentálása bonyolult, nehéz eligazodni a felhasználónak (TrustRadius, 2021).

A Ranorex kifinomult megoldásokat kínál a felhasználói felület elemeinek azonosítására és kezelésére amellett, hogy fejlett platformmal szolgál már több éve. A szoftver felhasználóbarát, azaz a felülete grafikus felvétellel segíti a szkript generálást. Előnyei az eszköznek, hogy részletes segítségnyújtással rendelkezik és nem igényel különösebb fejlesztői hozzáértést (felhasználóbarát). Rendelkezik külön csak futtatásra képes eszközzel, natív CI integrációval és több platformot támogat (webes, asztali, mobil). Fontos megemlíteni, hogy részletes a dokumentáció riportja. Negatív tényezői viszont, hogy csak egyben az összesplatformra (asztali, web, mobil) lehetséges csak licencet vásárolni, nem érhetőek el külön, így egy asztali alkalmazás teszteléséhez plusz költséget kell fizetnünk. Programnyelvei csak C# és VB.NET, ez korlátozott a többi automatizált szoftverhez képest, és az újabb frissítések gyakran tartalmaznak kompatibilitási hibákat, ezért nem annyira megbízható eszköz (Impact QA, 2020).

A TestArchitect széleskörű megoldásokat kínál minden platformra, rendelkezik DevExpress és CI támogatással. A támogatott programnyelvei a következők: C#, Python, Java. Hátrányként megemlítenénk, hogy nincs külön tesztfuttató eszköze, hanem a teljes eszköz újbóli beszerzése szükséges csökkentett licenc tartalommal, valamint nem lehet csak asztali tesztelő eszközt sem vásárolni, hanem a teljes licenc megvásárlása szükséges. Mindkét hatás nagyobb licenc-díjat eredményez a cég számára.

A TestComplete előnyei, hogy teljes mértékben támogatja a DevExpress megoldásokat, jól kidolgozott dokumentációval rendelkezik, a futtatás eredményriportja pedig érthető, könnyen emészthető. Natív CI integráció jellemzi, mely elengedhetetlen a folyamatos futtatásokhoz, valamint a program használatához nem szükséges mély fejlesztői vagy szakértői ismeret. Ez is díjköteles, viszont a különböző platformokat szeparáltan kezeli, így csak arra kell beruházni, ami a cég számára szükséges. Később a licenc bővíthető, ha új platformmal bővülne a cég szolgáltatási köre. A szoftver képpel is tudja illusztrálni a tesztelt eseményeket.

Összegezve az látható, hogy funkcionálisan az FX Software igényeit a Ranorex, a TestArchitect és a TestComplete tudja kielégíteni. Mindhárom rendszer tartalmaz teszt rögzítése-visszajátszási lehetőséget, a tesztírók számára a munkát megkönnyítő funkciókat, mindegyik képeket is készít a teszt során és könnyen értelmezhető és feldolgozható eredményriportot állítanak elő. A döntés végül az FX Software esetében a TestComplete-re esett, mivel rugalmasabb, a licencek vásárlása igény szerinti és megbízhatóbb is: évek óta az egyik legjobbnak értékelt program a felhasználói értékelések alapján.

3.2. TestComplete (TC) bemutatása

Kicsit részletesebben tekintsük át a TestComplete (továbbiakban TC) funkcióit és azok használatát.

A TC egységtesztre és a funkcionalitás tesztelésére is alkalmas, valamint a napi regressziós tesztekre is kiváló támogatást nyújt. A tesztek a programban két féle módon jöhetnek létre: rögzíti őket a szoftver vagy a tesztelők megírják a parancsokat. A tesztek a programban futnak, de van mód külső alkalmazás használatára is. A megadott alkalmazásokban felismeri az objektumokat és vezérlőket, ezért azzal kell kezdeni egy tesztelést, hogy ezeket hozzáadjuk a TestComplete-hez, ennek a neve NameMapping. Előnye, hogy speciális ellenőrző pontokat tartalmaz a rendszer, így a tesztet könnyebben tudjuk monitorozni. Számos beépített eszköze segít nekünk a műveletek végrehajtásában, továbbá a teszt megírásához egyéb segítségeket is ad azzal, hogy hozzáférést enged a szoftver belső objektumaihoz, metódusaihoz.

A következő platformokat támogatja (SmartBear, 2021):

Asztali alkalmazása Microsoft által támogatottak:

- valamennyi .Net fordítót, mint Jscript,
- Visual C#,
- Visual Basic,
- .NET,
- Delphi,
- Sybase PowerBuilder .NET 12
- Webes alkalmazásai:
- A 32 és 64 bites Microsoft Edge
- Microsoft Internet Explorer 11,
- Google Chrome 95,
- Mozilla Firefox 78 ESR.

Mobil alkalmazásai:

- Android és IOS egyaránt használható,
- Android 4.0.1-től 11.0-ig,
- IOS 13.0-15.1.

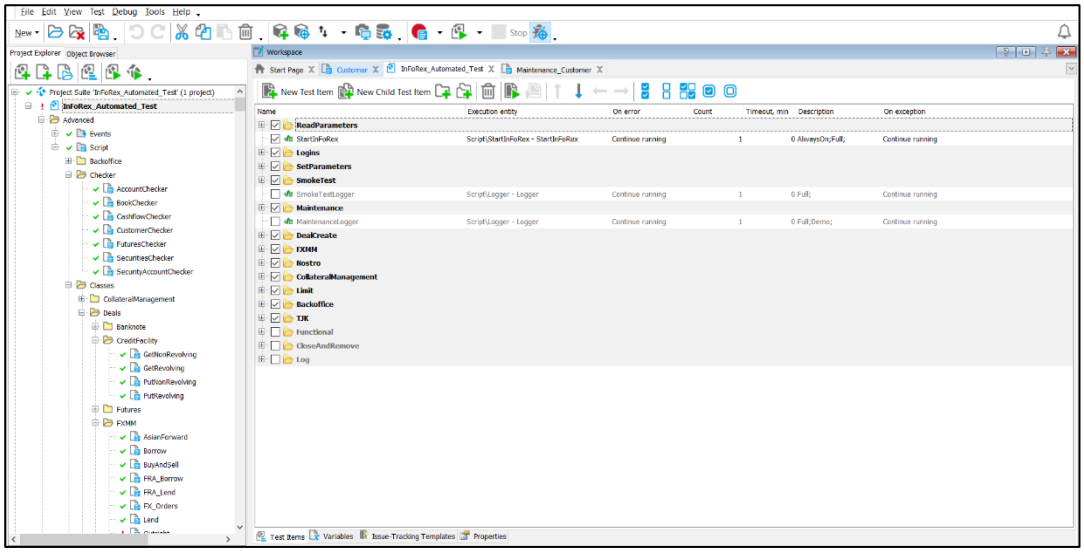
A TC rengeteg segítséget nyújt, hogy megkönnyítse a tesztek írását és futtatását. Továbbá a nyitó felület alapján látványosan felhasználóbarát (*1. ábra*). A főképernyő letisztult, egyszerű.

Mint a legtöbb programban, felül láthatjuk a menüsávot. Itt tudjuk állítani, hogyan nézzen ki a felület, és minden általános beállítás itt szerepel. Alatta találjuk az ikonokat, amik egyszerűsítik a munkánkat. Bal oldalt láthatóak a megírt szkriptek, illetve ezek alatt a futtatott tesztek eredményei találhatók. Jobb oldalon, középen jelenik meg a főablakunk, ahol dolgozunk. Lehet mappákba csoportosítani a szkripteket saját elképzelés szerint, és azt is beállíthatjuk, melyeket futtassuk egyszerre vagy külön-külön.

Számos hasznos funkció található a szoftverben, például el tudjuk menteni mind az ideiglenes, mind az ismétlődő változókat. Az ideiglenes változó esetén a változóknak adhatunk nevet, típust és alapértéket. Miközben fut a teszt, a változókon tudunk változtatni, amint vége a tesztnek ezek visszaállnak az alapértékekre. Az

ismétlődő változóknak hasonló tulajdonságaik vannak, de a fő különbség az, hogy a teszt futtatása közben, ha változnak az értékek, akkor a következő futtatásnál ezek nem állnak vissza alapértékre, hanem megtartja ezeket a változtatásokat (SmartBear, 2021).

1. ábra: **TestComplete nyitó felület**



Forrás: saját szerkesztés az InFoRex szoftver segítségével.

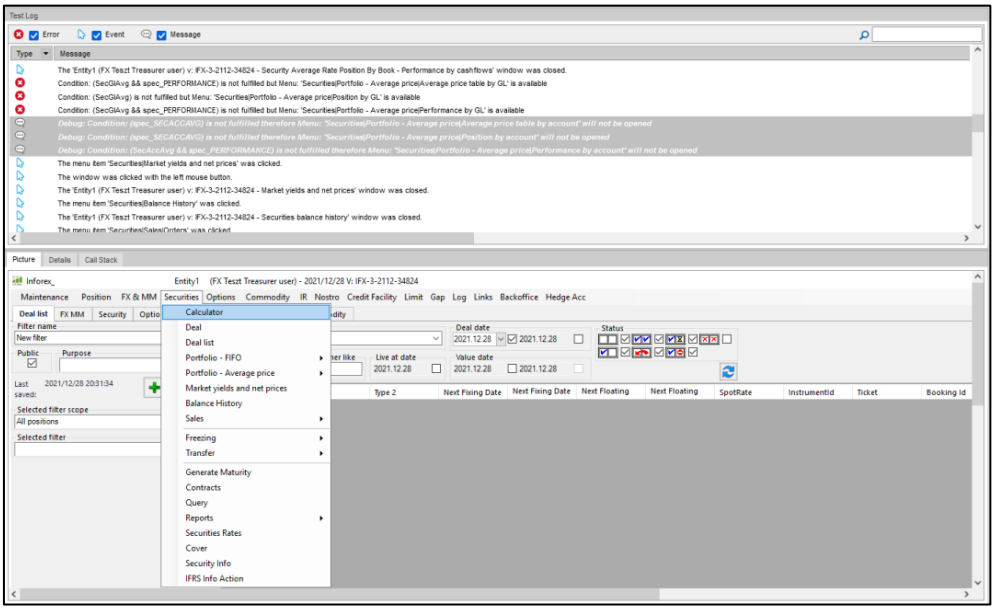
A kezdő programozóknak, szoftverfejlesztőknek a TC legkedvezőbb része a Record Script. Teszteket lehet írni úgy, hogy nem gépelünk egy sort sem. Manuálisan teszten végighaladva a TC kódsorba rögzíti azt. Hasznosítható ez a NameMapping-nél (a program egyes objektumaihoz egyedi név rendelhető) is, mert tulajdonképpen olyan, mintha egyesével végig néznék az objektumokat, csak rögzítjük is azokat, ezáltal megjegyzi az egyes elemeket, és így sok időt spórolhatunk. Célszerű ezzel kezdeni és nem egyesével menteni az objektumokat. A hiányos objektumokat előnyösebb az Object Spy-jal (segítségével az objektumokat és tulajdonságaikat tudjuk vizsgálni) felvinni.

Már meglevő és új szkriptekbe is el tudjuk menteni a rögzített tesztet, valamint vissza tudjuk játszani az elmentett szkripteket. Az egyes lépésekről külön-külön képernyőfelvétel készül, így látható, hogy mikor melyik lépésnél tart a tesztfolyamat. Az eredményt a Log fogja kiadni (*2. ábra*). A sorokként látszik az adott időben futott folyamat és annak időtartama. Bármelyik sor dupla kattintása ahhoz a kódsorhoz vezet, amelyet futtatott. Hibák esetén a TC több lehetőséget nyújt az okok feltáráshoz (például melyik szkript melyik sora hívta meg a hibára futott részt).

A TC futtatásához két megoldást használunk. Lokálisan is indíthatjuk őket a TC segítségével: kisebb rész-tesztek végzésekor, kódrészletet javítása után, vagy egy kisebb modul megírása közbeni ellenőrző tesztek esetén vesszük igénybe. Egyszerűbb a vizsgálata a tesztnek, mintha az egész rendszert tesztelnénk, mert kevesebb időt vesz igénybe, részletekben tekintjük meg a teszteredményeket,

továbbá az elkészült logot közvetlenül TC-ben kapjuk meg és elemezhetjük is a modult.

2. ábra: **A log**



Forrás: saját szerkesztés az InFoRex szoftver segítségével.

A másik variáció, az erőforrás-kímélő TestExecute segédprogram. A TC által létrehozott tesztek futtatását segíti elő azokon a gépeken, amelyekre nincs feltelepítve a program. Kizárólag csak teszteket képes futtatni, ezért a cég saját fejlesztésű előtét programmal (TestSetup) egészítette ki a működését. A TestSetup kitűnően alkalmazható külső virtuális gépeken tesztek indítására, bármilyen előzetes TC ismeret nélkül is. Előzetes paraméterezéseket és ellenőrzéseket végez a kiválasztott verzión, egy könnyen kezelhető felület segítségével. Ezt a fajta tesztelést akkor érdemes alkalmazni, ha egy átfogó teszt kivitelezése a cél. Pár órától több óráig is futhatnak a tesztek, függően a tesztek mennyiségétől, az implementáció nagyságától és a szoftver sebességétől. Előnyös éjszaka futtatni a teszteket, így általában másnap reggelre megkapjuk az eredményeket, de dolgozhatunk a teszt futtatással párhuzamosan is.

A TC-ben használt szkripteket összekapcsoltuk a SmartGit-tel. Ez egy verziókezelő szoftver, amely a különböző projekteket és szkripteket központilag tárolja, valamint segítségével egy tesztelő a saját munkaállomásán történő változtatásokat vagy hibajavításokat közzé tudja tenni a többi tesztelő számára. Látható mikor, mit és mire módosított egy fejlesztő. Ilyen módon gyorsan és szinkronban lehet haladni a problémák megoldásával, ezenfelül nyomon követhető a fejlődés.

Ha módosítás után a teszt rosszabbul végződött, mint amilyen állapotban volt eredetileg, látható melyik javítás volt a hiba okozója és vissza lehet állni az eredeti állapotba új megoldást keresve. A tesztelők korábbi javításaikból ötleteket is lehet

meríteni. Végezetül lehetőséget ad a SmartGit a verziókezelésre a TC-ben is, ahogy a rendszerből is különböző verziók szerepelnek kint a felhasználóknál, így elérhető, hogy párhuzamosan mind az ügyfél aktuális verziója, mind a továbbfejlesztett verzió tesztelhető legyen automatikusan.

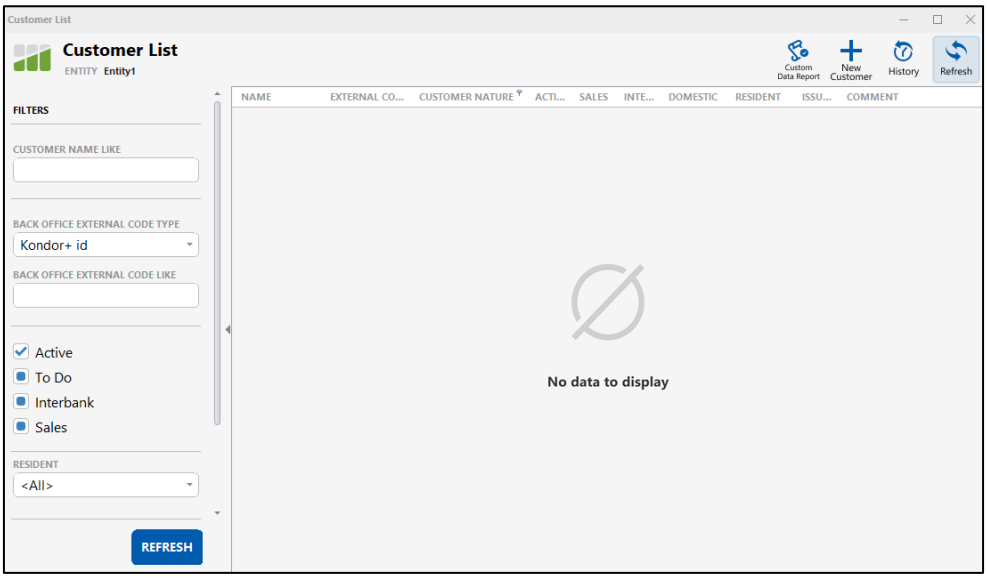
4. Az InFoRex rendszer tesztelése manuálisan és automatikusan

A következőkben bemutatjuk, hogy gyakorlatban hogyan történik az InFoRex tesztelése. Manuálisan és automatizáltan fogjuk ugyanazt a tesztesetet létrehozni, összehasonlítjuk a két módszert és megvizsgáljuk, hogy a gyakorlatban az *1. táblázatban* említett előnyök és hátrányok milyen mértékben érvényesülnek. Az InFoRex ügyfél (Customer) létrehozását teszteljük a példában, kitérve ennek időigényére, problémák kezelésére és az eredmény leírására. (Az InFoRex Customer felületei ügyfél implementációk szerinti paraméterezhető tartalommal jelenik meg, a demonstráció kedvéért egy kisebb paraméterezettségű, egyszerűbb implementációt használtunk fel.) Minden manuális és automata teszt egy előzetesen megírt tesztforgatókönyv és felhasználói leírás mentén történik.

4.1. Manuális tesztelés végrehajtása

Új ügyfél felvétele: új ügyfél regisztrációja az Ügyféllista felületről kezdeményezhető, a képernyő jobb felső sarkában található Új ügyfél gomb megnyomásával (*3. ábra*).

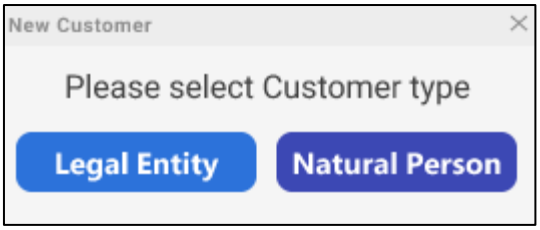
3. ábra: **Ügyfél felület**



Forrás: saját szerkesztés az InFoRex szoftver segítségével.

Először egy ügyféltípust kell kiválasztani a következő ablakban *(4. ábra)*, a példában kiválasztjuk a Natural Person-t, azaz a természetes személy opciót.

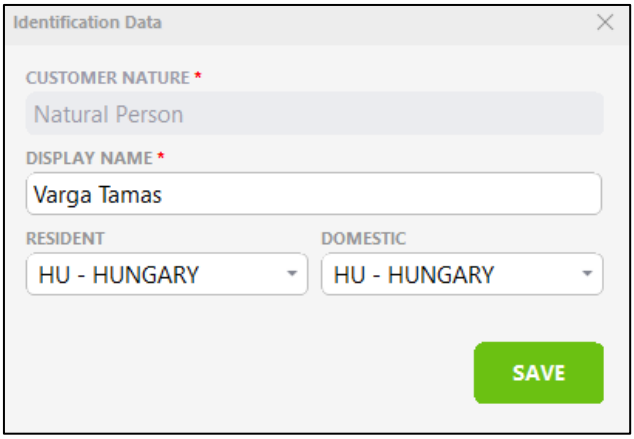
4. ábra: **Új ügyfél létrehozás**



Forrás: saját szerkesztés az InFoRex szoftver segítségével.

A következő ablakban *(5. ábra)* beírjuk az elsődleges rendszerbeli azonosítókat: jelen esetben csak az ügyfél nevét és a nemzetiségét (szerepelhetne itt például a személyi igazolvány szám, adószám, vagy vállalat esetén LEI kód, adószám stb.).

5. ábra: **Természetes személy adatok**



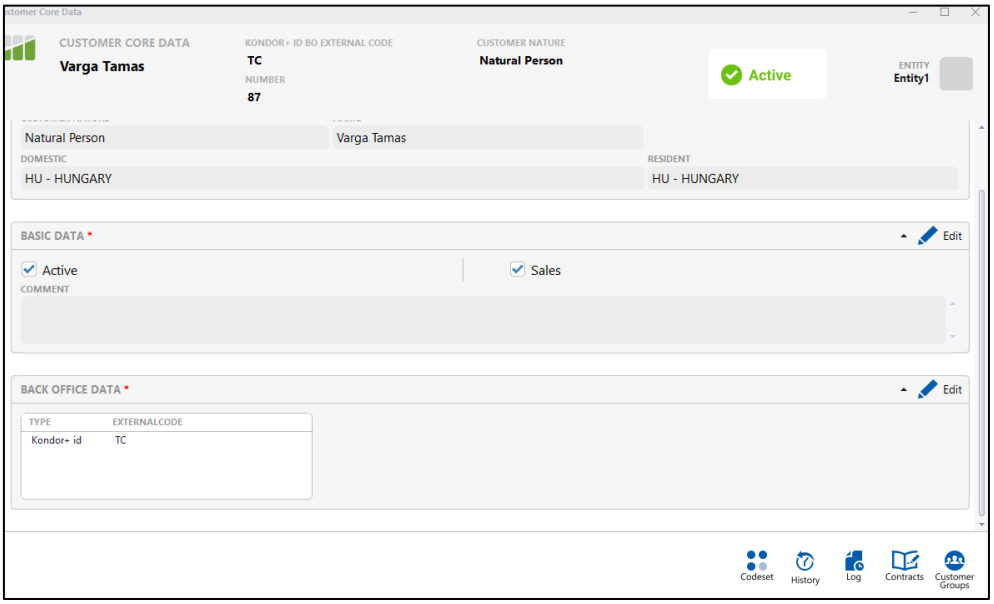
Forrás: saját szerkesztés az InFoRex szoftver segítségével.

A mentés gomb hatására felnyíló képernyőn (*6. ábra*) az ügyfél adatai különböző adatkörökben vannak elrendezve, ahol a meghatározott funkciókhoz vagy logikailag összekapcsolódó információk találhatók az egyes adatkörökben. Az adatkörök önállóan módosíthatók, de az egy körhöz tartozó adatok csak egyszerre szerkeszthetők, vagy (a négy szem elve esetén) elfogadhatóak.

Ügyfél rögzítése vagy egy adatkör módosítása után az ügyfél állapota „Módosítás alatt” értékre változik, és a rögzített vagy módosított adatkészlet jobb felső sarkában (az „Edit” gomb helyett) megjelenik az elfogadási és elutasító ikon, jelezve, hogy a második felhasználó elfogadhatja vagy elutasíthatja a módosított adatokat. Négy szem elvű elfogadási folyamat hiányában ez a lépés kimarad. A különböző adatköröket a második felhasználónak külön-külön kell elfogadnia, és az ügyfél „Módosítás alatt” állapotban marad, amíg a kötelező adatokat tartalmazó összes adatkészletet jóvá nem hagyják. Végül el kell fogadni az azonosító adatcsoportot, mivel az érvényesítési folyamat nem teszi lehetővé annak jóváhagyását, amíg a többi kötelező elemű kör át nem ment a négy szem elvű folyamaton. Tehát az ügyfél regisztrációs folyamat a többi adatkör kitöltésével

folytatódhat, ami az adatkörök jobb felső sarkában található szerkesztés (Edit) gomb megnyomásával lehetséges.

6. ábra: **Ügyfél adatok**



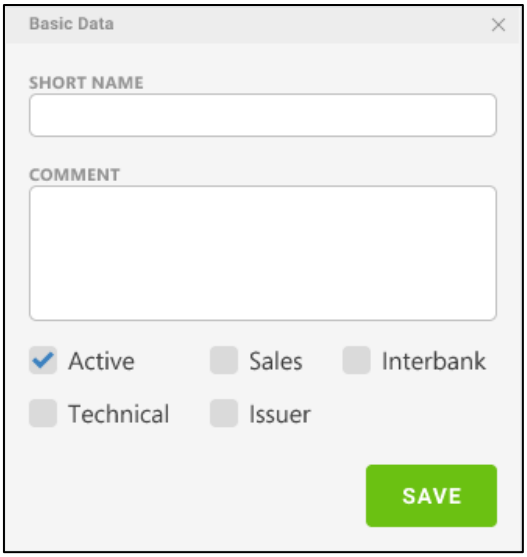
Forrás: saját szerkesztés az InFoRex szoftver segítségével.

A „Basic Data” lapon (*7. ábra*) a felhasználó beírhatja az ügyfél rövid nevét, aktiválhatja vagy inaktiválhatja a meglévő ügyfelet. Beállíthatja az ügyfél szerepkörét (sales, interbank, technical, issuer) és további megjegyzés is hozzáadható egy többsoros szöveges beviteli mezőbe. A mentés után a felhasználó visszakerül az összefoglaló képernyőre, ahol a módosított mezők sárga színnel kerülnek kiemelésre.

A Back Office adatkészlet elsősorban számviteli és könyvelési célokhoz szükséges információk gyűjteményeként szolgál, például külső azonosítók (például SAP-azonosító vagy Eurobank-azonosító), különböző ügyfélbesorolásokat támogató szektorkódok és statisztikai besorolásokhoz nélkülözhetetlen azonosítók megadására (*8. ábra*).

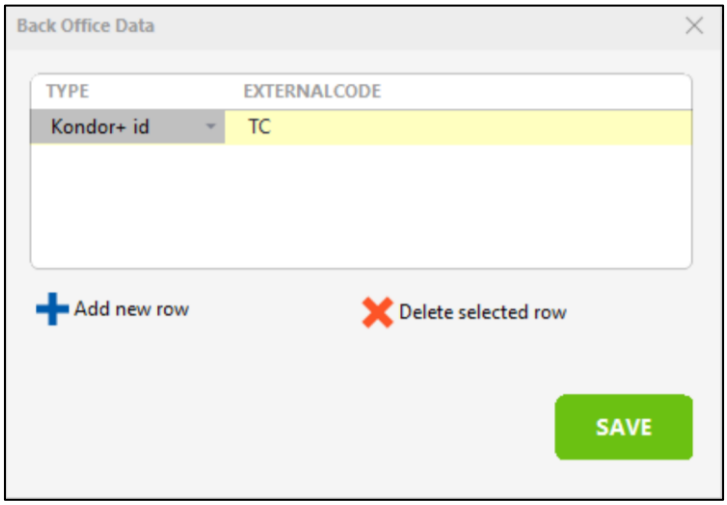
Minden szükséges információ feltöltése és mentése esetén az ügyfél aktív státuszra vált. Az utolsó lépésben ellenőrizzük, hogy a megadott ügyfél szerepel-e az adatbázisban, ha nem szerepel, akkor sikertelen a teszt, ennek eredményét jelentjük. Ezt úgy tudjuk megtenni, hogy először a „CUSTOMER NAME LIKE” mezőbe beírjuk az általunk létrehozott felhasználót. Kiválasztjuk a jelölőnégyzetekben, hogy milyen tulajdonságok szerint szűrjön a felület, végül a frissítés gombbal frissítjük az adatokat. Sikeres ügyfél létrehozás esetén megtaláljuk a keresett értékek között (*9. ábra*).

7. ábra: **Alap adatok**



Forrás: saját szerkesztés az InFoRex szoftver segítségével.

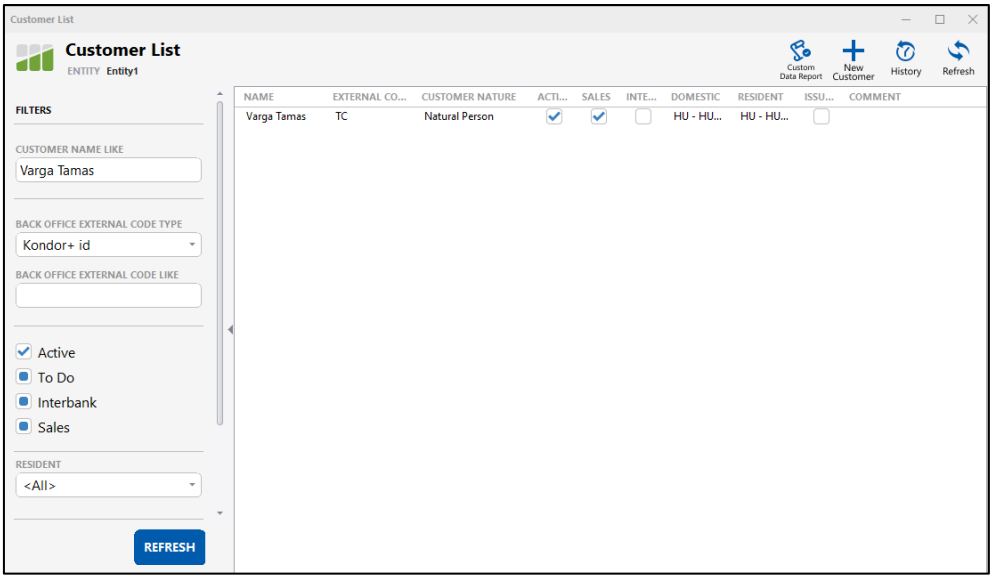
8. ábra: **Back Office adatok**



Forrás: saját szerkesztés az InFoRex szoftver segítségével.

A manuális teszt azzal zárul, hogy részletes jelentést készítünk a vizsgálatról, sikeres és sikertelen teszt esetén is elemezni kell a lépéseket a válaszidőkre és a pontosságra koncentrálva, a tesztjegyzőkönyv alapján.

9. ábra: **ellenőrzés**



Forrás: saját szerkesztés az InFoRex szoftver segítségével.

4.2. Automatizált tesztelés végrehajtása

Automatizált teszt folyamata megegyezik a manuális folyamattal, de a teszt kódolása révén. A kódban a könnyebb érthetőség kedvéért az egyes részek fölé írtuk, hogy melyik lépésben, mi következik. A teszt futtatását és az eredmény riportot is a gép fogja készíteni. A TestComplete szkripteket JavaScript-ben készítjük, jelen tesztrész parancssorát az *1. számú melléklet* tartalmazza. Fontos megemlíteni, hogy a példában bemutatott kód, nem alkalmas a többszöri futtatásra, ezt a gyakorlati használatban statikus nevek helyett változókkal és ellenőrzések beépítésével érjük el.

4.3. Manuális és automatizált tesztelés értékelése

Manuális tesztek közül egy fekete doboz tesztet hajtottunk végre, funkcionális, specifikáció alapú módon. Az FX Software által alkalmazott tesztelési folyamat megegyezik a V-modellel: a szakértők dokumentálják és elkészítik a tesztjegyzőkönyvet, ami tartalmazza a funkciók működését és tesztelhetőségét. Majd kézzel elvégezzük a manuális tesztet és a TC-ben megírjuk a kódot, lehetőség szerint már a fejlesztési folyamat során.

Az esettanulmányban bemutatott manuális teszt elvégzése egy rutinosabb tesztelőnek körülbelül egy percig tart. Kevésbé rutinos tesztelőnek beletelhet öt percbe is, mivel folyamatosan figyelnie kell a soron következő lépéseket. A teszt befejezésével a végeredményről leírást kell készíteni (hosszabb, összetettebb teszt megkívánja a teszt közbeni dokumentálást is). A teszt kezdete előtt nagyjából öt percet vesz igénybe a jegyzőkönyv megismerése. A teszt előrehaladtával a tesztelő megállhat ellenőrizni a lépések helyes sorrendjét. A sikeres eredményről nem feltétlenül szükséges részletes leírást készíteni, de ha valahol nem a jegyzőkönyv szerinti elvárt eredményt kapjuk, akkor a hibát rögzíteni, majd továbbítani kell a

fejlesztőknek. Összességében a tesztelő gyakorlata alapján 5-10 percet vesz igénybe ennek a manuális tesztnek az elvégzése.

Automatizált teszt elkészítése esetén több lehetőségünk is van a teszt megírására. Egy átlagos TC fejlesztő tizenöt-húsz perc alatt hozza létre a példában szereplő szkriptet, viszont a jegyzőkönyv megismerése ez esetben is nélkülözhetetlen. Kezdőként ajánlott a Record Scriptet használni, amely sok segítséget nyújt, például egy kód alapot, amelyet csak a megfelelő logikai felépítésre kell formálni. A kód megírásához használt sablon tesztforma használata lehetővé teszi a további felhasználást. A kód megírását követően lefuttattuk lokálisan a tesztet az esetleges hibák kiszűrése érdekében, majd ellenőriztük az elkészült log eredményt is. Sikeres validáció esetén, a kódrészlet felkerül a központi teszt szkriptek közé további felhasználásra. A bemutatott példa automatizáltan harminc-negyvenöt másodperc alatt fut le, függően az éppen használt adatbázis nagyságától és az adott verzió állapotától. Összesen 20-30 perc alatt elkészíthető egy megfelelő TC szkript ehhez a tesztesethez, amely aztán újra felhasználható.

5. Összegzés

A szoftver tesztelése elengedhetetlen része a szoftverfejlesztésnek. A tesztelési fázis legfontosabb funkciója a kritikus hibák kiszűrése, amelyek nem kerülhetnek a végfelhasználók elé. A manuális tesztelési módszer mellett már innovatív automatikus tesztelési technika is bevezethető a tesztelési folyamatba. A tanulmányban összehasonlító módszertannal bemutattuk, mely szempontok szerint célszerű automatizált tesztelői programot választani. A felállított előzetes követelményrendszerünk alapján, mint például az eredmények riportálhatósága vagy a tanulhatóság, öt programot hasonlítottunk össze. A választás a TestComplete programra esett, mivel az alapvető technológiai és gazdasági követelményeken felül további előnyös tulajdonságokkal is bír, mint a megbízhatóság vagy a rugalmasság.

Egy esettanulmányon keresztül rávilágítottunk arra is, hogy a manuális vagy az automatizált szoftvertesztelés a hatékonyabb tesztelési forma és milyen esetekben van szükség manuális tesztelésre. A manuális és automatizált tesztelés összehasonlításának konklúziója az elvártnak megfelelő: egyszeri alkalom esetén a manuális tesztelés gyorsabban elvégezhető, ad hoc tesztelés esetén hatékonyabb tesztelési módszernek bizonyult. Ismétlődő tesztelés esetén azonban, mint a regressziós tesztelés, már pár ismétlés után (a példában a tesztelő rendszerismerete vagy TC ismerete alapján 2-6-szoros idő alatt) az automatizált tesztelés a hatékonyabb megoldás. Többször futtatható, stabilabb tesztek készítéséhez több idő szükséges, de például egy legalább tíz ügyféllel rendelkező szoftvergyártónak (mint például az FX Software) már egyetlen átfogó tesztelési kör alatt is megéri az automatizált tesztelés irányába mozdulni. Segítségével magasabb színvonalú szoftver juthat el a végfelhasználókhoz, kevesebb tesztelési munkát és így több, más termelő tevékenységre fordítható időt eredményez a cégek számára.

Köszönetnyilvánítás

Jelen cikk a 00116-os számú projekt részeként, az Innovációs és Technológiai Minisztérium Nemzeti Kutatási Fejlesztési és Innovációs Alapból nyújtott támogatásával, a 2020-1.1.2-PIACI-KFI-2020 pályázati program finanszírozásában valósult meg.

Irodalomjegyzék

AppLabs (2008): Test Automation: Delivering Business Value. <https://web.archive.org/web/20100106191031/http://www.applabs.com/internal/app_whitepaper_test_automation_delivering_business_value_1v00.pdf> (2021.10.03.)

Catelani, M., Ciani, L., Scarano, V. L., Bacioccola, A. (2011): Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use. *Computer Standards & Interfaces*, 33 (2): 152–158. https://doi.org/10.1016/j.csi.2010.06.006

Diaz, E., Tuya, J., Blanco, R. (2003): Automated software testing using a metaheuristic technique based on tabu search. Proceedings of 18th IEEE International Conference on Automated Software Engineering, (2003): 310–313. https://doi.org/10.1109/ASE.2003.1240327

Impact QA, Shah, M. (2020): Ranorex: Pros & Cons of GUI Test Automation Tools. <https://www.impactqa.com/blog/ranorex-pros-cons-of-gui-test-automation-tools/> (2021.11.12.)

ISTQB (2021): Szoftvertesztelés. <http://istqb.org/> (2021.11.15.)

Javatpoint (2020): Software Testing. <https://www.javatpoint.com/manual-testing> (2021.11.03.)

Microsoft (2021): Use Coded UI tests to test your code. <https://docs.microsoft.com/en-us/visualstudio/test/use-ui-automation-to-test-your-code?view=vs-2022&viewFallbackFrom=vs-2022https%3A%2F%2Fwww.trustradius.com%2Fproducts%2Fappium%2Freviews%3Fqs%3Dpros-and-cons> (2021.11.12.)

Myers, G. J., Badgett, T., Sandler, C. (2011): *The Art of Software Testing*. Third Edition. Wiley, https://doi.org/10.1002/9781119202486

SmartBear (2021): Testcomplete Documentation. <https://support.smartbear.com/testcomplete/docs/general-info/introducing-testcomplete.html> (2021.11.12.)

STF (2020): Sofware Testing Fundamentals. <https://softwaretestingfundamentals.com/> (2021.11.2.)

Testim, Phil Voullet (2019): What Is Test Automation? A Simple, Clear Introduction. <https://www.testim.io/blog/what-is-test-automation/> (2021.11.10.)

Tompa T. (2019): Szoftvertesztelés Szoftverfejlesztési modellek. <https://users.iit.uni-miskolc.hu/~tompa/Szoftverteszt/2_Modellek.pdf> (2021.11.03.)

TrustRadius (2019): Appium Reviews. <https://www.trustradius.com/products/appium/reviews?qs=pros-and-cons> (2021.11.12.)

1. számú melléklet

Automatizált tesztelés parancssora

```
function NaturalCustomerCreate()
{
//Customer felület megnyitása
Aliases.InFoRex.frmMain.Activate();
let inFoRex = Aliases.InFoRex_BO;
let frmMain = inFoRex.frmMain;
frmMain.StripMainMenu.Click("Maintenance|Customers|Customer List");

//NaturalCustomer létrehozás kezdete
inFoRex.CustomerListView.headerPanel.btnAddNew.ClickButton();

//Natural Person kiválasztás
inFoRex.layoutControl.btnNaturalPerson.ClickButton();

//Név Beírása
inFoRex.NaturalPersonEditView.dataLayoutControl.txtName.TextBoxMaskBox.Click();
inFoRex.NaturalPersonEditView.dataLayoutControl.txtName.SetText("Varga Tamas");

//Hely kiválasztás
inFoRex.NaturalPersonEditView.dataLayoutControl.cmbResidentCountryId.TextBoxMaskBox.Click();
inFoRex.NaturalPersonEditView.dataLayoutControl.cmbResidentCountryId.SetText("HU - HUNGARY");
inFoRex.NaturalPersonEditView.dataLayoutControl.cmbDomesticCountryId.TextBoxMaskBox.Click();
inFoRex.NaturalPersonEditView.dataLayoutControl.cmbDomesticCountryId.SetText("HU - HUNGARY");

//Tulajdonságok megadása
let CustomerShowView = inFoRex.CustomerShowView;
CustomerShowView.layoutControlBlocks.basicShowView.dataLayoutControl.Items.Item(0).CustomHeaderButtons.Item(0).RaiseCl
ick();
inFoRex.BasicEditView.dataLayoutControl.chkIsSales.Click(10,13);
inFoRex.BasicEditView.dataLayoutControl.btnSave.ClickButton();

//BackOffice adatok megadása
let CustomerShowBOView = inFoRex.CustomerShowView;
CustomerShowBOView.layoutControlBlocks.backOfficeShowView.dataLayoutControl.ItemsItem(0).CustomHeaderButtons.Item(0).R
aiseClick();
let BackOfficeEditView = inFoRex.BackOfficeEditView;
BackOfficeEditView.dataLayoutControl.btnAddExternalCode.ClickButton();
DevExpClickCellFunction(BackOfficeEditView.dataLayoutControl.grdExternalCode, "EXTERNALCODE", "");
inFoRex.TextEdit.SetText("TC");
BackOfficeEditView.Close();

//Customer elfogadás
let CustomerShowView_2 = inFoRex.CustomerShowView_2
CustomerShowView_2.layoutControlBlocks.naturalPersonShowView.dataLayoutControl.Items.Item(0).CustomHeaderButtons.Item(
0).RaiseClick();
CustomerShowView_2.Close();

//Ellenőrzés
inFoRex.CustomerListView.splitContainerControl.SplitGroupPanel2.dataLayoutControlFilter.txtNameLike("Varga Tamas");
inFoRex.CustomerListView.splitContainerControl.SplitGroupPanel2.layoutControlRefresh.btnRefreshInFilte.ClickButton();

//Segédeszköz
 }
  function DevExpClickCellFunction (Grid, Column, Value)
{
var NewGrid, RowIndex;
NewGrid = Grid;
var NewColumn = Column;
var NewValue = Value;
RowIndex = DevExpFindRowFunction (NewGrid, NewColumn, NewValue);
Log.Message("RowIndex: " + RowIndex);
if (RowIndex >= 0)
{
  NewGrid.ClickCell(RowIndex, NewColumn);
  NewGrid.DblClickCell(RowIndex,NewColumn);
  return true;
}
else
{
  return false;
 }
}
```