SZAKDOLGOZAT

Hóbel Tímea

Debrecen

2011

Debreceni Egyetem

Informatikai Kar

Tesztelés ipari környezetben

Belső témavezető:

Dr. Kósa Márk Szabolcs

Egyetemi tanársegéd

Külső témavezető:

Boda Béla

Neuron Szoftver Kft.

Készítette:

Hóbel Tímea

Programtervező Informatikus

Debrecen

2011

Tartalomjegyzék

1. Bevezetés

A szakdolgozatom témaválasztása igen nagy fejtörést okozott számomra mindaddig, amíg el nem kerültem a Neuron Szoftver Kft-hez egy kötelező szakmai tantárgy gyakorlati keretein belül. A gyakorlat teljesítése után megpályáztam a cégnél hirdetett szoftvertesztelői pozíciót, amelynek elnyerésének köszönhetően elkezdtem megismerkedni a szoftverek tesztelésével, a tesztelési fázissal. Így szakdolgozatomban a szoftverfejlesztési folyamat e fontos fázisát, a tesztelést fogom elemezni. Bemutatok különböző tesztelési módszereket, és néhány olyan eszközt, amivel megkönnyíthető, valamint esetenként meggyorsítható ez a folyamat. Különös figyelmet kap a Selenium keretrendszer, amelynek felépítéséről részletesen írok, illetve egy gyakorlati példán keresztül is szemléltetem a működését, és bemutatom, hogyan használható tesztek automatizálásához. Manapság kevés figyelmet kap ez a terület, de egyre nyilvánvalóbbá válik a folyamat fontossága. A szoftverek implementálásakor, és fejlesztésekor még a gyakorlott programozók is ejtenek hibákat, átlagosan 8-10 soronként egyet, amiket minél előbb, és minél hatékonyabb módszerekkel kell feltárni majd javítani.

Legideálisabb esetben létezik egy tesztelő csoport, akik az adott szakterület szakembereinek tekinthetők, azonban a legtöbb cégnél nincs ilyen szakembergárda. Ha a feltételek megfelelőek, és mégis van annyi munkatárs, hogy el lehet különíteni olyan munkacsoportot, akik a teszteléssel foglalkoznak, akkor ezeket többnyire kezdő informatikusok alkotják. Ezek miatt tartom fontosnak, és hatékonynak a tesztrobotok tesztelésbe történő bevonását, mert kiegészíthetik, vagy bizonyos esetekben helyettesíthetik az informatikusok munkáját. Dolgozatomban arra szeretnék rávilágítani, hogy hogyan teszi hatékonyabbá a tesztelési folyamatot az, ha a robotok végzik a főbb funkciók ellenőrzését – akár munkaidőn túl –, és a munkatársak erőforrásai a speciális esetek feltárására, és problémák kijavítására összpontosíthatóak.

2. A szoftverfejlesztés folyamata

2.1 Szoftverfolyamat

A szoftver termék előállítására irányuló tevékenységek sora a szoftverfolyamat. Az általános tevékenységek:

- igényfelmérés: a felhasználó igényeinek, követelményeinek felmérése, rendszerezése;
- specifikáció: a szoftver feladatainak, és a megszorításoknak specifikációja;
- tervezés, és implementáció: a szoftver nagyvonalú, és részletes tervének kidolgozása, programozás, egységtesztelés;
- integráció: a szoftver részeinek összeállítása, és tesztelése;
- verifikáció, és validáció: annak bizonyítása, hogy az elkészített szoftver a követelményeknek megfelelően működik, és a felhasználói igényeket is kielégíti;
- evolúció: a szoftver karbantartása, továbbfejlesztése a folyamatosan változó igényeknek megfelelően.

2.1.1 A követelmények feltárása

A megrendelő, felhasználó igényeinek megismerése, rendszerezése, és specifikálása, ami szakterületi ismereteket igényel. Fontos a követelmények teljessége, és következetessége, ugyanis sok esetben csak a rendszer átadásakor derül ki, ha néhány követelményt kifelejtettünk vagy félreértettünk.

2.1.2 Specifikáció

A követelmények következetes, és teljes leírása. A szoftver által megvalósítandó feladatok meghatározása, és rendszerbe foglalása. Azt tartalmazza, hogy mit csináljon a szoftver, és nem azt, hogy hogyan. A követelmények folyamatosan változnak, ezért a specifikációnak is változtathatónak kell lennie. Ez befolyásolja a szoftverfolyamat további fázisait is. A szoftverrel szemben a követelmények változásának igénye sokkal gyakoribb, mint a hagyományos termékek esetében.

2.1.3 Tervezés, és implementáció

A tervezés, és implementáció a specifikáció futtatható programmá konvertálása. A szoftver tervezése:

- a szoftver, és az adatok struktúrájának meghatározása, modellek készítése;
- a szoftverkomponensek közti kapcsolatok, interfészek megtervezése;
- a komponensek tervezése;
- az adatszerkezet, és az algoritmusok tervezése.

Az implementáció a programozást, és az egységteszteket jelenti.

2.1.4 Integráció, validáció, verifikáció

A szoftver részeinek összeillesztése, együttműködésük tesztelése, és az elkészült szoftver beillesztése környezetébe, a környezeti kapcsolatok tesztelése. A tesztelésre számos stratégia létezik, választásuk az alkalmazott technológiától függ. A validáció, és verifikáció olyan ellenőrző, és elemző folyamatok, amelyek biztosítják, hogy a szoftver megfelel a specifikációnak, és kielégíti a megrendelő igényeit. A verifikáció azt vizsgálja, hogy a szoftver megfelel-e a specifikációjának, míg a validáció azt vizsgálja, hogy a felhasználó számára megfelelő-e a szoftver. Verifikálni, és validálni nem csak a kész terméket kell, hanem minden fejlesztési fázisban, vagyis a követelményeket, és a terveket is. Egy szoftver elfogadási szintjét befolyásolja:

- a funkciója, vagyis hogy mire akarjuk használni;
- elvárások a szoftverrel szemben;
- piackényszerítő ereje arra készteti a cégeket, hogy minél hamarabb kész legyen a szoftver, ennek következménye a rengeteg béta verzió.

2.1.5 Evolúció

Az alkalmazásba vett szoftvert használat közben változtatni kell:

- a használat közben felfedezett hibák javítása miatt;
- új funkciók beépítése, vagy meglévő funkciók változtatása miatt;
- a környezet változása miatt.

A szoftvert már a tervezéskor fel kell készíteni a karbantartásra, továbbfejlesztésre. Csak a részletesen, jól dokumentált szoftver alkalmas a karbantartásra, és a továbbfejlesztésre. A szoftver jövője nem függhet attól, hogy a fejlesztők emlékeznek-e még az évekkel korábbi döntések, megoldások indokaira.

3. A tesztelési fázis

3.1 Tesztelés, és belövés

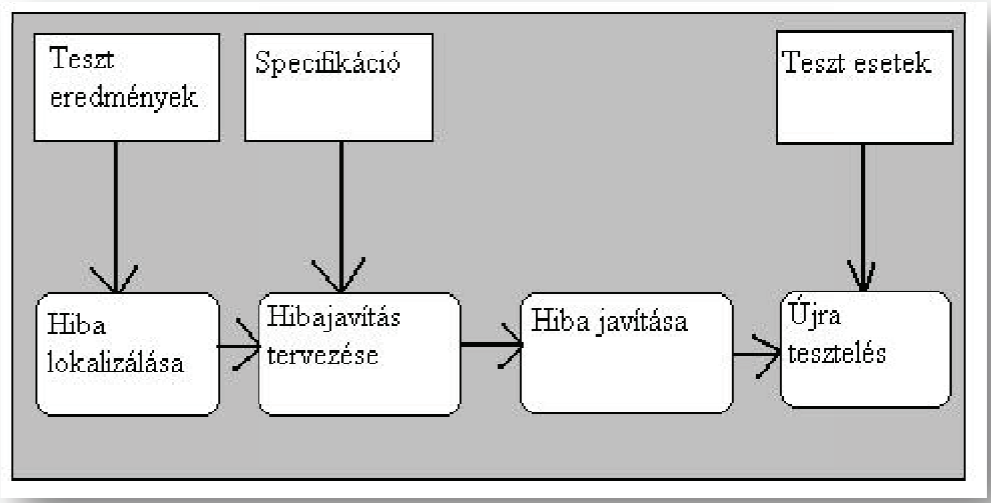
A validáció, és verifikáció folyamaton belül a rendszer ellenőrzésére két technika használható. Az egyik egy statikus technika, melyek alatt a szoftverátvizsgálásokat, és automatikus elemzéseket értjük. A másik pedig egy dinamikus technika a tesztelés. A V&V folyamat során kiderülnek a program hiányosságai, így a programot módosítani kell a hibák kiküszöbölésének érdekében, ezt nevezzük belövési folyamatnak, melyet további V&V tevékenységekkel ötvözhetünk. A belövés, és a V&V különálló folyamatok, melyeket nem kell integrálni:

- a V&V annak megállapítására alkalmas, hogy a rendszerben vannak-e hiányosságok, hibák;
- a belövés behatárolja, és kijavítja ezeket a hiányosságokat.

Nincsen egyszerű, standard módszer a belövésre, a hibakeresők mintákat keresnek a teszt kimenetében, és a hiányosság típusának a programozási nyelvnek, a programozási folyamatnak az ismeretében próbálják beazonosítani a hibát. Ismerik a tipikus programozási hibákat, a programozási nyelvre jellemző hibákat, és azokat illesztik a megfigyelt mintákra.

3.2 Tesztelés, és hibakeresés

A verifikáció, és validáció feladata a programhibák jelenlétének feltárása, a hibakeresés pedig ezen hibák lokalizálásával, és javításával foglalkozik. A hibák megtalálása érdekében a hibakeresés során a program viselkedéséről hipotéziseket állítunk fel, amiket ellenőrzünk.



A hibakeresés folyamata

A tesztelési, és vizsgálati eljárások sikere érdekében körültekintő tervezésre van szükség. A tervezést már a fejlesztés korai fázisában el kell kezdeni. A terv határozza meg a statikus verifikálás, és a tesztelés egyensúlyát. A teszt tervezése a tesztelési eljárás irányelveit fogalmazza meg, nem kell a terméktesztelést itt leírni.

A szoftvertesztelési terv struktúrája:

- A tesztelő eljárás: a tesztelési eljárás főbb fázisainak leírása.
- Követelmények követhetősége: a teszteket úgy kell megtervezni, hogy minden követelményt külön lehessen tesztelni.
- Tesztelt elemek: a fejlesztési eljárás azon elemeit specifikáljuk, amelyeket tesztelni kell.
- A tesztelés menetrendje: a tesztelés menetrendje a szükséges erőforrások foglalásával. Természetesen szorosan kapcsolódik a teljes projekt ütemezéséhez.
- A tesztek rögzítésének eljárása: a teszteket nemcsak futtatni kell, hanem az eredményeket szisztematikusan rögzíteni is. A tesztelési eljárásnak felülvizsgálhatónak kell lenni.
- Hardver, és szoftver szükségletek: a szükséges szoftver eszközök listája a becsült hardver használattal együtt.
- Kényszerek: a tesztelési eljárást befolyásoló kényszerek, például munkaerőhiány.

3.3 A tesztelés folyamata

A tesztelés egy dinamikus technika, mely során az elkészült rendszert működés közben vizsgáljuk. Ezeket a rendszerteszteket az adott szakterület, illetve adott technológia legtapasztaltabb embereinek kellene végezniük, ezzel szemben sok cég friss diplomásokat alkalmaz tesztelésre. A tesztelés két fajtája ismert:

- hiányosságtesztelés: a program, és a specifikációja között meglévő ellentmondások felderítése. Az ilyen tesztek a rendszer hiányosságainak feltárására tervezik, nem a valós működés szimulálására.
- statisztikai vagy validációs tesztelés: a program teljesítményének, és megbízhatóságának tesztelése valós körülményeket szimulálva, vagyis a teszteseteknek a valós felhasználói bemeneteket, és azok gyakoriságát kell mutatnia.

Másik megközelítésben komponens-, és rendszertesztelésről beszélhetünk. A komponenstesztelésnél függvényeket, objektumokat vagy újrafelhasználható komponenseket tesztelünk. Ezeket utána alrendszerbe vagy a teljes rendszerbe integráljuk, és azt nézzük meg, hogy az így kapott rendszer megfelelő funkcionális, és nem funkcionális tulajdonságokkal rendelkezik-e. A komponensek tesztelése során a tisztán azonosítható komponensek funkcionalitásainak tesztelése történik, amit általában a programozók végeznek, de kritikus rendszereknél külön csapat végzi. Az integrációs tesztelés alatt pedig a komponensek közötti interakciók, és a teljes rendszerre vonatkozó funkcionalitás tesztelését értjük. Előhozhat komponenseken belüli hibákat, mely az előző folyamat megismétlését eredményezi. Különálló csapat végzi ezt a tesztelést egy részletes, írott specifikáció alapján.

4. Tesztelési technikák

4.1 Egység teszt, modul teszt

Az implementáció során minden elkészült egységről el kell dönteni, hogy önmagában megfelelően működik-e. Tulajdonképpen a konkrét kódban szereplő függvényeket, alprogramokat, rutinokat teszteljük abból a célból, hogy megmutassuk, hogy a program egyes részei külön-külön, önállóan is jól működnek. Ezt a folyamatot elsősorban a fejlesztők, programozók végzik, azok az informatikus, akik az adott implementációért felelősek, nem

pedig azok a személyek, akik a szoftver tesztelésével foglalkoznak. Általában automatizált tesztesetekkel dolgoznak ebben a szakaszban, amit pontosan meg kell tervezni, mivel a rendszer további változásakor minden egyes verzió esetén újra kell majd futtatni. Így elsőre feleslegesnek tűnhet ezt elvégezni, de ha jobban belegondolunk, akkor ezzel a tesztelési szakasszal már a fejlesztési szakaszban felderíthetők, és ami még fontosabb, javíthatók a hibák. Ez legvégső soron alacsonyabb költséghez vezet, ugyanis minél később tárunk fel egy hibát, annál költségesebb annak javítása. A tesztelésnek ezt a fajtáját szokás white-box tesztelésnek is nevezni, mivel eközben belelátunk a szoftver belső működésébe.

4.1.1 Izolációs tesztelés

A modulok egymástól elszigetelten kerülnek tesztelésre. Az egységek tetszőleges sorrendben tesztelhetők. Nagy előnye, hogy párhuzamosan tesztelhetőek az egységek, illetve a módosítások csak az adott modul változását vonják maguk után. Hátránya viszont, hogy nem biztosítják az egység korai integrációját.

4.1.2 Top-down tesztelés

A hierarchia legfelső szintjén álló elem teszteléséhez az eggyel lejjebb álló elemek viselkedését, és interfészét szimuláló ideiglenes elemek szükségesek. Ha a teszt sikeres, akkor az ideiglenes elemeket a valódiakkal helyettesítjük, és az általuk használtakat pedig újabb ideiglenes elemekkel szimuláljuk. Előnyei közé tartozik, hogy jól illeszkedik a top-down programfejlesztési módszerekhez, egy modul megírása után rögtön tesztelhető, az esetleges tervezési hibák korán kiderülnek, és idejében orvosolhatók, viszonylag korán rendelkezésre áll egy korlátozott képességű rendszer, ami a validációt könnyíti meg. Hátránya, hogy bonyolult lehet a szimulációt végző ideiglenes rutinok megírása illetve a hierarchia felső szintjein álló modulok sokszor nem szolgáltatnak outputot.

4.1.3 Bottom-up tesztelés

Először a legalsó szinten lévő modulokat teszteljük, majd a hierarchiában felfelé haladunk. Ehhez a felső szinteket szimuláló tesztelés környezetet kell írni. A módszer előnyei, és hátrányai fordítva jelentkeznek, mint az előbb említett top-down tesztelés esetén.

4.1.4 Útvonal tesztelés

Útvonaltesztelés: fehér-doboz tesztelési stratégia, amelynek az a célja, hogy a programban minden független végrehajtási útvonalat kipróbáljunk. Ebben az esetben minden utasítás legalább egyszer lefutott, és minden feltétel tesztelt igaz, és hamis esetre is. A programban található független utak száma általában arányos a program méretével, ezért az útvonal tesztelést az egység-és modulteszt nél használják. Az útvonal tesztelés kezdete egy programfolyamat-gráf. Ha a program goto mentes, akkor ezt könnyű előállítani. A szekvenciális utasítások (értékadás, eljáráshívás…) kihagyhatók a gráfból. A feltételes utasítások minden ága külön út lesz a gráfban, a ciklusok pedig a ciklusfeltételt reprezentáló pontba visszamutató nyilak. Egy ilyen gráfban a független utak száma meghatározható a folyamatgráf ciklomatikus komplexitásának kiszámításával (McCabe, 1976). CG(G) = élek száma – csomópontok száma + 2. Azon programoknak, amelyek nem tartalmaznak goto-t, a ciklomatikus komplexitása eggyel több, mint a programban található feltételek száma. Feltétel alatt itt egyszerű logikai feltételt értünk. Ha egy logikai kifejezés tartalmaz logikai operátort, akkor az már összetett feltételnek számít, ami annyi darab egyszerű feltételnek felel meg, ahány ilyen feltételt kapcsoltunk össze benne logikai operátorral. Ennek a strukturális tesztelési módszernek az a célja, hogy bizonyítsa, hogy minden független programútvonalon végigmentünk. Egy független programútvonal alatt egy olyan utat értünk, amely a folyamatgráfban legalább egy új csomóponton keresztülmegy. A valamennyi független programútvonal teszteléséhez minimum annyi teszteset szükséges, mint amennyi a gráf ciklomatikus komplexitása. Egyszerű programoknál nem nehéz a folyamatgráf felrajzolása, de amikor egy program nagyon összetett szerkezetű, akkor érdemes dinamikus programelemzőt használni a program futási profiljának felderítésére. Miután a programelemző lefutott, a futási profil megmutatja, hogy a program mely részei futottak, illetve maradtak ki a tesztesetek végrehajtása során.

4.2 Alrendszer, és rendszer teszt

Egy nagy rendszer általában feloszthető egymástól függetlenül implementálható részrendszerekre. Egy részrendszeren, alrendszeren belül az azt alkotó modulok közötti együttműködés tesztelhető. A rendszer teszt egyik célja az alrendszerek esetleges hibás együttműködésének kiderítése. Másik fontos feladata, hogy a rendszer teljesíti-e a vele

szemben támasztott, és a követelmény analízisben lefektetett funkcionális, és nem funkcionális követelményeket.

4.2.1 Integrációs tesztelés

Alrendszerek összeépítését teszteli. Tipikus hiányosság tesztelés. Regressziós teszten alapul. Történhet:

- Lentről felfelé: a modulok összeépítéséből kapjuk a rendszert. Teszt meghajtók szükségesek, melyen a környezet viselkedését szimulálhatjuk. Akkor használjuk, ha az architekturális terv később alakul ki. Nehezebb a teljes rendszer működését demonstrálni;
- Fentről lefelé: az egész rendszert teszteljük. A magas szintű elemek tesztelésével kell kezdeni. Bizonyos funkciók még nem implementáltak mikor tesztelünk, ezért szimulálni kell őket, ami időnként problémás.

4.2.2 Kiadási tesztelés

Valójában tekinthető validációs tesztelésnek. Ha a felhasználót is bevonjuk a tesztelésbe, akkor elfogadási tesztelés. Itt az ún. fekete doboz tesztet alkalmazzuk. Nem ismerjük a kódot, inputokat adunk neki, és figyeljük az outputot. Fekete doboz tesztelés esetén a rendszer belseje nem ismert, viselkedése csak a bemenetekre adott válaszok alapján érthető meg. Itt a tesztek a program vagy komponensspecifikációból származnak. Az input adatok megtervezésénél figyelembe kell venni a következőket:

- a rendellenes inputokat vissza kell dobnia a rendszernek;
- ki kell deríteni a puffer túlcsordulást okozó inputot;
- ugyanazt az inputot többször is ki kell próbálni;
- minden inputnak vagy egyfajta tartománya, és ezen tartományok szélsőséges adataira is tesztelni kell.

A szoftvereknél egy új kiadás előállításánál a kiadási tesztet a cég végzi. És, ha egy adott szinten megbízhatónak minősítik a rendszert, akkor azt mondják, hogy előállt egy alfa verzió. Ekkor jön a béta teszt, amibe már külső embereket is bevonnak. Ezután áll elő egy olyan kiadás, amiben a hibák számát lecsökkentették. Ezek után jelenik meg a szoftver, adott megbízhatósági szinttel.

4.2.3 Stressz tesztelés

Ez egy teljesítményteszt. A rendszert megpróbáljuk egyre nagyobb tesztelésnek kitenni. Két irányba mehet el. Egyre nagyobb adatokkal tesztelünk, vagy a rendszerrel való interakciók számát növeljük. A normális rendszer a stressz tesztelés hatására is működőképes marad. Folyamán kimérhető a működési profil. Az a rendszer működik jól, ami hibatűrő, nincs olyan inputja, amitől elszáll, és hangolható. A működési profil a hangoláshoz fontos, pl. tartalmazza azt az információt, hogy a program mely funkciókat használja sokszor. Ezeknek a funkcióknak kell nagyon jól működniük. Ma problémát jelent, hogy a rendszerek kis része hangolható.

4.2.4 Smoke teszt

A hibák felderítésének leghatékonyabb módja, legtöbbször automatizált tesztekkel végzik. A szoftver főbb funkcióinak működésnek ellenőrzésére használják. A szoftveren történő különböző verzióváltáskor érdemes használni, ugyanis egy verzióváltáskor történő javítás vagy fejlesztés okozhat hibát az eddig jól működő alapvető funkciókban, ezért egy smoke teszt futtatása minden egyes verzióváltást követően ajánlott. Mivel ezek az alapvető funkciók nagy valószínűséggel ritkán vagy egyáltalán nem változnak, ezért nagyon könnyű automatizált tesztet írni, és azt végrehajtani a verzióváltások kiadása előtt.

4.2.5 Regressziós tesztelés

Újra lefuttatjuk a tesztet ugyanazokra az input adatokra, amelyek korábban hibát okoztak, ezzel bizonyítva, hogy a javítás sikeres volt. Bármilyen szoftvertesztelés lehet regressziós tesztelés, ha regressziós hibák feltárására hivatott. Regressziós hibáról beszélünk, ha egy korábban működő szoftver funkcionalitás hibásan vagy egyáltalán nem működik, tipikusan olyan hibák, amelyek javítások nem várt következményeként jelentkeznek.

A hiba lehet:

- lokális: változtatások miatt új hibák keletkeznek;
- leleplező: a változtatás egy már korábban is létező, de nem jelentkező hibát fed fel;
- távoli: az egyik részben végzett változtatás egy teljesen más részben hoz elő hibát.

Említésként néhány regressziós tesztelési típus:

- régi hibák újratesztelése: akkor hajtjuk végre ezt a tesztelést, amikor a régi hibákat kijavították, és azt akarjuk igazolni, hogy a hibák megszüntetése nem tökéletes;
- mellékhatás regressziós tesztelés: a lényeges részek újratesztelése, annak igazolása érdekében, hogy a korábban jól működő programrészek a javítás után már nem működnek jól;
- explorativ tesztelés: elvárjuk a tesztelőtől, hogy ismerje a korábbi tesztek során felmerülő problémákat, és az újabb tesztek így már hatékonyabban lesznek, mert a tesztelő növekvő tapasztalatain alapul.

4.3 Elfogadási teszt

Célja a rendszer elfogadtatása, annak bizonyítása, hogy a rendszer a felhasználó igényeinek megfelel. Ehhez nem szimulált tesztadatok, hanem valós adatok, és valós környezet, és ha rendelkezésre áll, akkor valós felhasználó szükséges. Ez a tesztelési fázis kimutatja, hogy a követelmény analízis hiányos volt, illetve az elkészült rendszer nem felel meg minden hatékonysági követelménynek. Az elfogadási teszt szokásos elnevezése még az alfa teszt, a rendszer pedig az alfa verzió. Ha a rendszer egyedi megrendelésre készül, akkor ebben a fázisban általában a leendő felhasználó használja a rendszert, és gyakran a betanítás, illetve a bevezetés kezdeti szakaszát is jelenti. A fázis végét az jelenti, ha a fejlesztők, és a felhasználók egyetértésre jutnak abban, hogy a rendszer éles használatra alkalmas.

5. Tesztdokumentumok

5.1 Tesztelési terv

Az üzleti követelményrendszert tartalmazó dokumentum szolgáltatja az alapot egy megvalósítási terv elkészítéséhez, ami alapján elkezdhetjük tervezni a teszteseteket. Ehhez legelső lépésként egy teszttervet kell készíteni, amelyben rögzítünk, és részletezünk a teszthez kapcsolódó elemeket, mint például magát a célt, amelyre a teszt irányul. A tervnek tartalmaznia kell a hozzá felhasznált, és közben létrejött dokumentumokat, a használni kívánt tesztelési módszereket, eszközöket, és a környezetet. Ezen események felelőseit, és a hozzá tartozó elfogadási kritériumokat, határidőket.

5.2 Teszt specifikáció

Ebben a dokumentumnak részletezzük, és fejtjük ki a teszttervben leírt módszereket, jellemzőket. Itt kerül definiálásra a tényleges teszteset lépésről-lépésre, a meghatározott feladat ellenőrzéséhez.

5.3 Tesztelési jelentés

A tesztek elvégzése után készül a tesztelési jelentés, amiben a tapasztalataink rögzítése mellett dokumentáljuk az elvégzett tesztek sikerességét vagy éppen sikertelenségét. Fontos előtte tisztázni, hogy mit tekintünk sikeres, illetve sikertelen tesztnek.

6. Selenium

6.1 A Seleniumról

A Selenium egy hordozható szoftvertesztelő keretrendszer webes alkalmazásokhoz. A tesztesetek számos programozási nyelven lehetőségünk van elkészíteni. A tesztek futtatását több böngészőben is elvégezhetjük. Részei:

- Selenium IDE;
- Selenium Remote Control;
- Selenium Core.

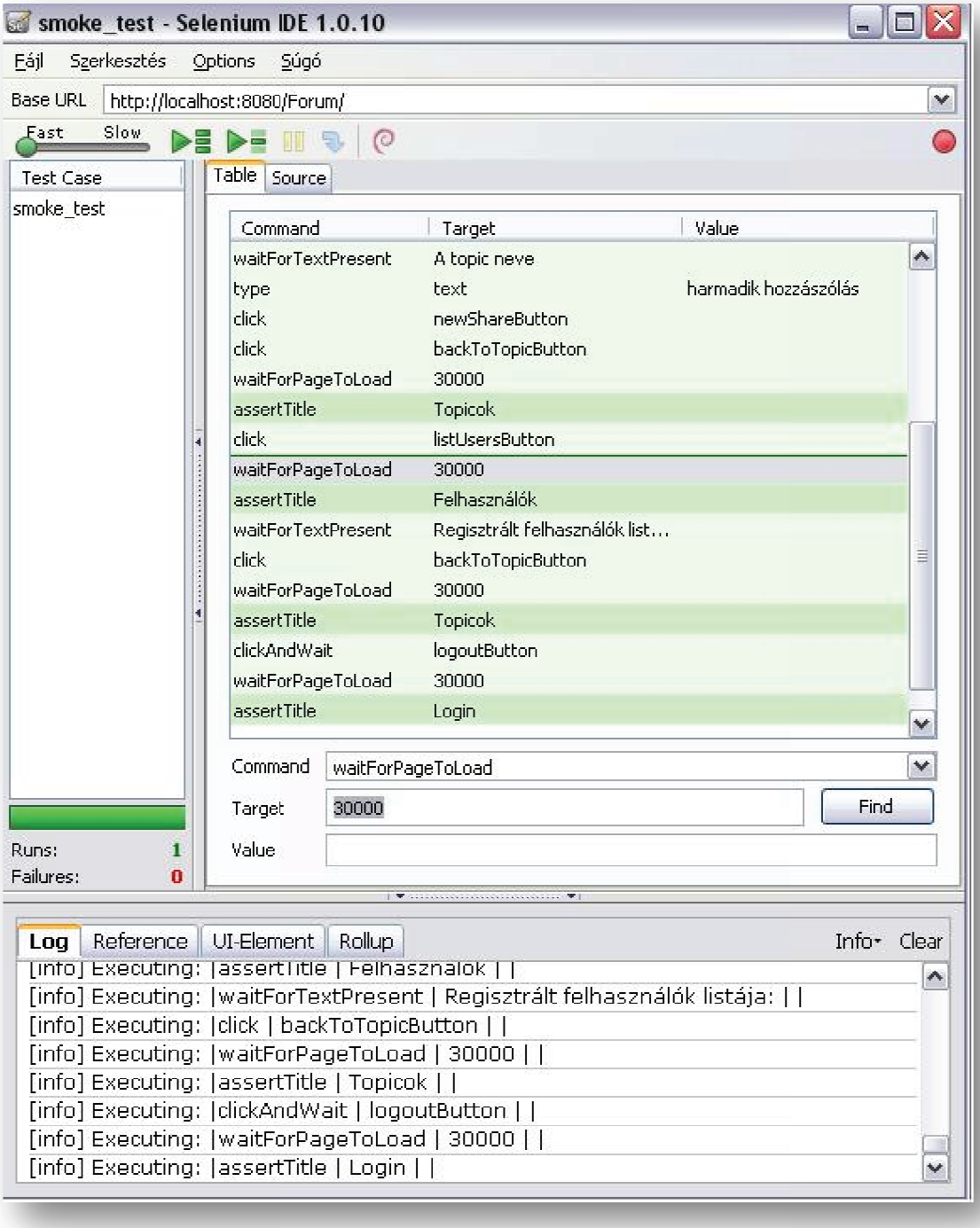
6.1.1 Selenium IDE

A Selenium IDE tulajdonképpen egy Firefox plugin, mellyel teszteseteket lehet rögzíteni, futtatni. Megkönnyíti számunkra a tesztesetek automatizálását, lehetőséget biztosít a tesztek könnyű rögzítésére, módosítására, és lejátszására, breakpointok segítségével történő debuggolásra. Beépített, hatékony funkciókat is tartalmaz, mint például az intelligens mezőfelismerés vagy a tesztek különböző nyelveken történő mentése. Rendelkezésünkre állnak különböző pluginok, amelyek installálásával bővíthető az IDE funkcionalitása. Példaként meg lehet említeni azt a plugint, amivel a teszteset közben eltárolt változók értékeit ki lehet íratni egy külön fülön. Az IDE legnagyobb hátránya, hogy csak a Firefox böngészőt támogatja, azon belül is csak Firefox 2-vel vagy magasabb verzióval alkalmazható. A

platformok között azonban nem tesz kivételt, egyaránt használható Windowsra, Linuxra vagy akár Solarisra telepítve.

6.1.1 Az IDE használata

Egy Firefox böngésző indítása után a http://seleniumhq.org/download/ oldalra navigálva letölthetjük a kiegészítőt, ami automatikusan feltelepül a böngészőbe. Indítása az Eszközök menüből történik, amit a következő oldalon található ábra szemléltet.



Selenium IDE

Az IDE indításakor megjelenő ablakban jobb oldalt láthat a Rec gomb  , ami alapértelmezetten bekapcsolt állapotban van. Ez arra szolgál, hogy a felületen történő eseményeket (gombok megnyomása, input mezők kitöltése, checkboxok, radiobuttonok aktiválása stb.) rögzíti. Ezt egy táblázatos formában jeleníti meg, aminek 3 oszlopa van, egy

Command, egy Target, és egy Value. A Command oszlop a végrehajtandó selenium parancs nevét tartalmazza. Az hogy melyik felületi elemen kell ezt a parancsot végrehajtani, azt a Target oszlopból olvashatjuk ki. A Value oszlopot pedig nem minden esetben kell kitölteni. Konkrét példákkal szemléltetve:

- ha egy felhasználónevet kell begépelni, akkor szükséges a value oszlop kitöltése:

Command	**Target**	**Value**
type	felhasználónév elérése	maga a felhasználónév

- ha rákattintunk egy linkre, akkor nincs szükség a value oszlop kitöltésére:

Command	**Target**	**Value**
click	a link azonosítója	

A rögzített parancsokat természetesen kiegészíthetjük kézileg berögzített utasításokkal, az esetek többségében szükséges a kiegészítés. A Source fülön tudjuk megnézni a forrásunkat HTML formátumban. Lehetőség van más programnyelvre átkonvertálni ezt a forrást, ezt az Options -> Format menüpont alatt tehetjük meg. A futtatás sebességét is állíthatjuk a Csúszka 

állításával. Majd ha berögzítettük a tesztet, akkor a Futtat gombbal  tudjuk futtatni, a lépések eredményeit pedig a Log fülön olvashatjuk. Lehetőség van tesztforgatókönyvek futtatására is. Létrehozhatunk test suite-t a már rögzített, és mentett test case-kből, amelyeket valamilyen sorrendben berögzítünk. A tesztesetek a tesztforgatókönyvben rögzített sorrendjében szekvenciálisan kerülnek futtatásra a másik Futtat gomb  hatására. Az IDE bal hasábjában olvasható a test case neve, illetve ha suite-ról beszélünk akkor a test case-ek nevei a megadott sorrendben. Ez alatt található 2 sor 

, amiben a futtatások eredményei láthatóak, az első sor a futtatott tesztek száma, a második pedig azt összegzi, hogy ezek közül hány volt olyan, ami nem volt sikeres.

6.1.2 Selenium Remote Control

Tesztelési eszköz, amely lehetővé teszi webes alkalmazások automatizált tesztelését. Megoldást nyújt nekünk olyan esetekben, amikor nem csupán egyszerű böngésző

manipulálására szolgáló parancsok futtatására van szükség. Selenium RC használja a programozási nyelvek teljes könyvtárszerkezetét, hogy még összetettebb teszteket lehessen készíteni. Olyan esetben használjuk a Selenium RC-t, ha a teszt összeállítása olyan logikát igényel, amit a Selenium IDE nem támogat. Ilyen lehet:

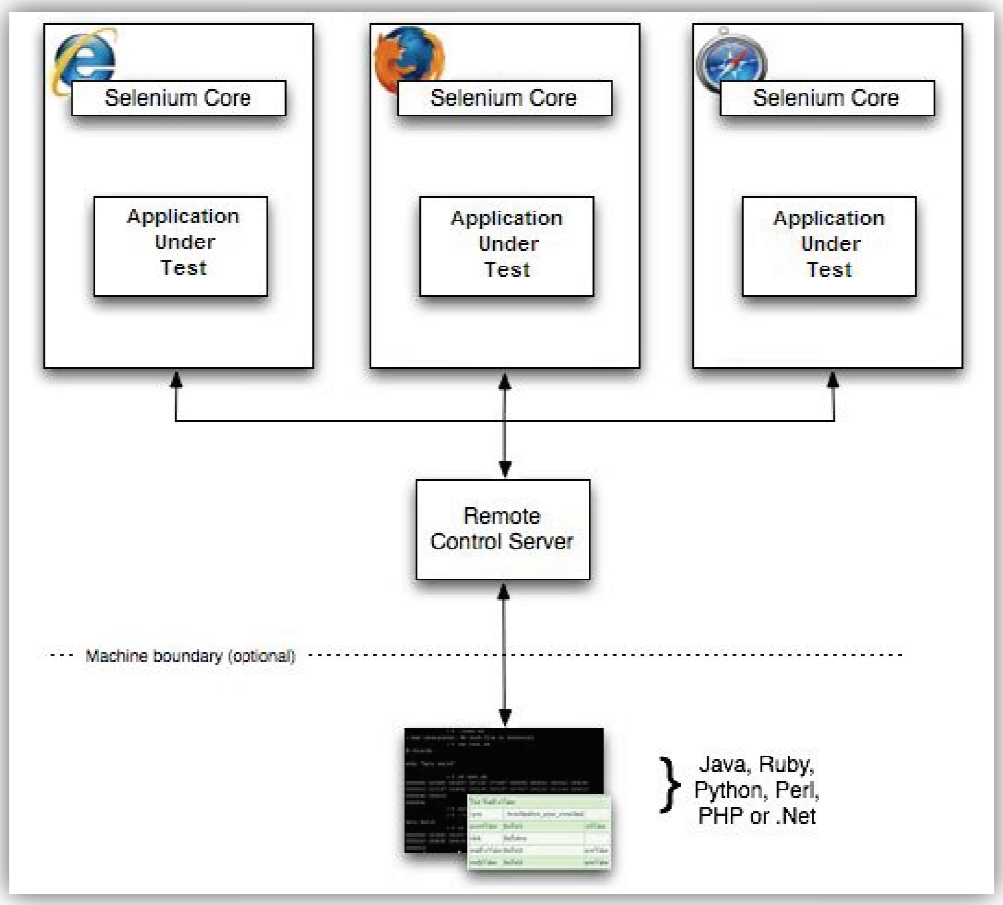
- teszteredmények naplózása, illetve jegyzőkönyvezése;
- hibakezelés, legfőképpen a nem várt hibák esetén;
- adatbázis tesztelés;
- teszteset csoportosítás;
- hibával leállt tesztek újrafuttatása;
- teszteset függőségek.

Selenium RC részei:

- Szerver;
- Kliens driver, a használt programozási nyelvvel.

Szerver

A szerver automatikusan elindítja, és leállítja a böngészőt. Feldolgozza a teszt program Selenium parancsait, és visszajelzést ad a kapott eredménnyel. Indítása parancssorból a következő paranccsal történik java –jar selenium –server.jar –interactive. A Remote Control szerver a Selenium Core csomagokból áll, és automatikusan betölti azokat a megadott böngészőbe. Működése a következő oldalon található ábrán figyelhető meg.



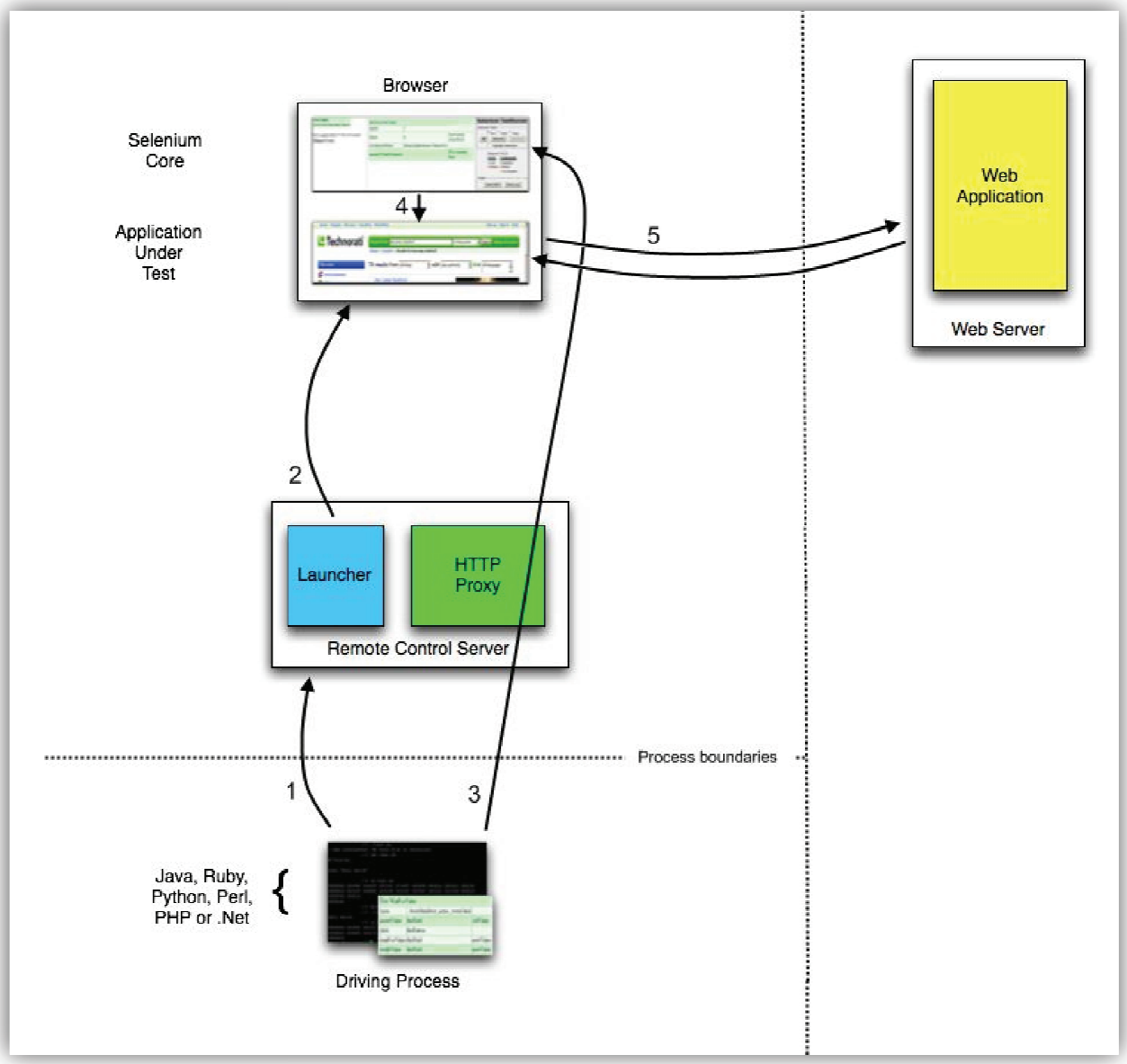
A Selenium RC architektúrális felépítése

Kliens driver

Egy interface-t ad meg az adott programozási nyelv, és a Selenium RC szerver között. Különböző programnyelvekre nyújt támogatást Selenium parancsok felhasználására.

Működése:

1. A kliens eléri a Selenium szervert;
2. A Selenium szerver elindít a böngészőben egy URL-t, ami betöltődik a Selenium Core weboldalán;
3. A Selenium Core megkapja az első utasítást, ami tipikusan a tesztelendő felület kezdőoldalának megnyitása;
4. A webszerver megkeresi ezt az oldalt, és megjeleníti egy ablakban.



A Selenium RC működése

6.1.3 Selenium Core

Webes alkalmazások tesztelésére alkalmas. A tesztek futtatása közvetlenül a böngészőben történik úgy, mintha egy valódi felhasználó ülne ott, és végezné el a tesztelés lépéseit. Mondhatni platformfüggetlen, mivel használható több féle operációs rendszer (Windows, Linux, Macintosh) alatt futtatott Internet Explorer, Firefox, Opera böngészőkben. Egy JavaScript/DHTML alapú nyílt forráskódú szoftver, ami ingyenesen letölthető, és használható.

6.2 Selenium API

A Selenium API egy keretrendszer, amely magában foglalja azokat a parancsokat, és utasításokat, amivel a selenium objektumunkat manipulálni tudjuk. Több csoportba sorolhatók:

- Műveletek (Actions): azokat az utasításokat soroljuk ide, amelyek közvetlenül befolyásolják az alkalmazás állapotát, mint például a „klikk az adott linkre”, vagy a „válaszd a megadott funkciót” stb. Ha egy ilyen művelet megbukik vagy hibát okoz, akkor az aktuális teszt futtatása befejeződik. Több művelet nevében szerepel egy úgynevezett „AndWait” utótag, ami annyit eredményez, hogy az adott utasítás végrehajtása után megadott ezredmásodpercnyi időt várakozik, amíg a szerveroldali kérés visszaérkezik, és az oldal újra betöltődik;
- Kiegészítők (Accessors): megvizsgálják az alkalmazás egy állapotát, és az eredményt egy változóban tárolják. Ilyen például a „storeTitle”, ami az adott oldal címét tárolja;
- Állítások (Assertions): hasonló a kiegészítőkhöz, de abban különbözik, hogy azt vizsgálja, hogy az alkalmazás egy állapota megfelel-e az elvártaknak. 3 csoportba oszthatók assert, verify, és waitFor:
    - egy assert vizsgálat: ha hamissal tér vissza, akkor a teszt megáll;
    - egy verify ellenőrzés: ha hamissal tér vissza, akkor a teszt futása folytatódik, és a hiba naplózásra kerül;
    - egy waitFor utasítás: vár bizonyos feltételek teljesülésére. Amint teljesülnek a feltételek a teszt futása folytatódik.

Egy Selenium parancs 3 részből áll, egy Command, egy Target, és egy Value részből. A Command a selenium utasítás neve, a Target az első paraméter, amit mindig kötelező megadni, míg a Value rész opcionális.

6.2.1 Elem meghatározók (Element Locators)

Mint azt az elnevezésük is elárulja, ezek azt a HTML elemet határozzák meg, amelyre az utasítások vonatkoznak. Formátuma:

locatorType = argument

Többféleképpen meghatározhatjuk ezeket a lokátorokat:

- id = id, válasszuk azt az elemet, amelynek az @id attribútuma id. Ha nem talál ilyet, akkor azt az elemet válasszuk, amelynek a @name attribútuma id;
- dom = javascriptExpression, a megadott sztring kiértékelése alapján keresi meg az elemet;
- xpath = xpathExpression: határozzuk meg az xpath kifejezés által megjelölt elemet;
- link = textPattern: válasszuk azt az elemet, amelyre a megadott minta illeszkedik;
- css = cssSelectorSyntax: válasszuk a szelektor által meghatározott elemet;
- ui = uiSpecifierString: egy UI specifikus sztringet határoz meg egy másik lokátorba, amit kiértékelve határozza meg az elemet.

Meghatározott lokátor előtagok nélkül a következőket vallja a Selenium:

- dom, azok a lokátorok, amelyek „document”-tel kezdődnek;
- xpath, azok a lokátorok, amelyek „//”-rel kezdődnek;
- identifier, egyébként.

Használhatók különböző szűrők, hogy a lokátorral megadott elemek számát szűkítsük. Ilyen szűrőelem lehet a:

- value = valuePattern, amely egy adott értékű elemre szűr, vagy az;
- index = index, amely egy megadott indexű elemre szűkít.

Alkalmazhatók még különböző mintaillesztési konvenciók a Value paraméter értékére:

- glob:pattern: válasszuk a mintára illeszkedő karaktersorozatot;
- regexp:regexp: a reguláris kifejezésnek eleget tevő karaktersorozatot válasszuk;
- regexpi:regexpi: az eszközfüggetlen reguláris kifejezést kielégítő karaktersorozatot válasszuk;
- exact:string: a meadott, konkrét sztringgel egyező karaktersorozatot válasszuk.

6.2.2 Néhány, a példában is használt selenium utasítás

- click(locator): egy adott elemre kattint. Lehet ez akár egy link, egy gomb, egy checkbox vagy akár egy rádiógomb. Ha a kattintás eredménye egy új oldalra navigál, akkor mindenképpen meg kell hívnunk a waitForPageToLoad utasítást, amely maximum az argumentumában megadott ezredmásodpercnyi időt vár az oldal betöltődéséig;

- open(url): megnyitja a paraméterben megadott URL címet;
- tpye(locator, value): a locator argumentumban hivatkozott mezőbe beírja a value paraméterben megadott tetszőleges sztringet;
- windowMaximize(): teljes képernyős módra váltja az ablakot.

7. Egy konkrét automatizált tesztelés bemutatása

7.1 A feladat megfogalmazása

Nem szerettem volna egy már meglévő alkalmazáson bemutatni a tesztelést, nagyobb kihívásnak tartottam, ha a tesztelendő alkalmazást is én magam készítem el. Ezért készült el a hallgatói fórum implementációja. A gyakorlatban a tesztelési fázishoz akkor jutunk el egy szoftver készítése során, ha előtte egy követelményspecifikációban megfogalmazott igényeknek megfelelően elkészül a fejlesztés vagy már vannak kész olyan funkciók, melyek működése a fejlesztési fázissal párhuzamosan is leellenőrizhető. A konkrét feladat az, hogy az elkészült hallgatói fórum alkalmazás funkcióit leteszteljük, és átadhatónak nyilvánítsuk a szoftvert. Amennyi tesztet csak lehet automatizált teszttel végezzünk. A folyamatot folyamatosan dokumentáljuk. Az átadást pedig egy olyan tesztjelentés fogja biztosítani, melyben a tapasztalataink összegzése mellett az összes teszteset sikeres futása áll.

7.2 A hallgatói fórum alkalmazás követelményspecifikációja

Ezt a dokumentumot megtekinthetjük az 1. számú függelékben.

7.3 A követelményspecifikáció alapján elkészült alkalmazás

Az előbbi pontban említett követelményspecifikációban leírt igények alapján a következő Java osztályok implementációja készült el:

- AddHozzaszolas.java
- AddNewFelhasznalo.java
- AddNewTopic.java
- Database.java
- Felhasznalo.java

- Hozzaszolas.java
- ListazFelhasznalok.java
- ListazHozzaszolasok.java
- ListazTopics.java
- Login.java
- Logout.java
- Topic.java

AddHozzaszolas.java

Új hozzászólások megosztásáért felel. Az alkalmazás Hozzászólások című oldalán hívódik meg, amikor a Megosztás gombra kattintunk. Hatására a témához tartozó összes eddigi, illetve az input mezőbe beírt új hozzászólás ki listázása történik. Az új hozzászólást elmenti az adott téma hozzászólásai közé.

AddNewFelhasznalo.java

Lehetőség van új felhasználó regisztrálására, ez az alkalmazás elindításakor a főoldalról a Regisztráció gombra kattintva érhető el. A regisztráció folyamán kötelező megadni egy nevet, ami a felhasználónév, illetve egy jelszót, opcionális lehetőségként kitölthető a születési dátum is. Amennyiben ez üresen marad, akkor egy 0 default értéket kap. Üresen nem maradhat sem a név, sem pedig a jelszó mező, illetve már korábban regisztrált felhasználónévvel megegyező nevet nem adhatunk meg a regisztráció során.

AddNewTopic.java

Új témák létrehozásáért felel. Az alkalmazás Témák című oldalán hívódik meg, amikor a Létrehozás gombra kattintunk. Az input mezőben kötelező valamilyen téma nevet megadni, üres mező esetén figyelmeztet minket, hogy nem töltöttük ki a mezőt. Akkor is figyelmeztetést kapunk, ha egy már létező téma nevével azonos nevű témát próbálunk meg létrehozni. Egyébként pedig létrehoz egy új témát, melynek neve a mezőben megadott sztring lesz, és az első hozzászólás is legenerálódik hozzá a következő formában.

- ${a téma létrehozója} létrehozta a(z) ${input mezőbe írt sztring} nevű témát!

Database.java

Témák, és felhasználók tárolására szolgál. Tartalmaz néhány statikusan beregisztrált felhasználót, témát, illetve hozzászólást. Ilyen felhasználók az admin, és a user, ilyen témák a Statisztika, és a Mesterséges intelligencia. Valamint a Statisztika téma tartalmaz a létrehozásakor automatikusan legenerálódott első hozzászóláson kívül egy másodikat, amit a user felhasználó írt, és a következő szöveget tartalmazza: Csatlakozom a topichoz, ezzel javítva a statisztikát :)!

Felhasznalo.java

A fórum felhasználóit implementáló osztály, egy nev, jelszo, és szuletesiEv adattaggal.

Hozzaszolas.java

A fórum hozzászólásait implementáló osztály, egy sorszam, topic, text, és szerzo adattaggal.

ListazFelhasznalok.java

A Database osztályban tárolt felhasználókat listázza ki.

ListazHozzaszolasok.java

A téma azonosítója alapján megkeresi a megfelelő témát a Database osztályban, majd paraméterként átadja a hozzátartozó ListazHozzaszolasok.jsp-nek, ami kiírja a megadott témához tartozó hozzászólásokat a felületre.

ListazTopics.java

Lekérdezi a Database osztályban tárolt témákat, és egy listában visszaadja a hozzátartozó ListazTopics.jsp-nek, ami kiírja őket a felületre.

Login.java

Lekérdezi a Database osztályban szereplő felhasználókat, és amennyiben talál egyezést a begépelt név, és jelszó mezőkkel, akkor az adott felhasználót paraméterben átadja, és belép a fórumba. Egyébként, ha nem töltjük ki vagy hibás adattal töltjük ki valamelyik mezőt, akkor azt jelzi.

Logout.java

A felhasználó paramétert null paraméterrel adja át, és visszanavigál a főoldalra, azaz kiléptet a fórumból.

Topic.java

A fórum témáit implementáló osztály, egy id, nev, szerzo adattaggal, és egy hozzaszolasokat tartalmazó listával.

Az implementáció még korántsem teljes, ugyanis a felületi megjelenítésekért felelős JSP osztályok implementációja még hiányzik. A fórumhoz tartozó osztályok ábécé sorrendben:

- AddNewFelhasznalo.jsp
- ListazFelhasznalok.jsp
- ListazHozzaszolasok.jsp
- ListazTopics.jsp
- LoginHiba.jsp
- RegHiba.jsp
- TopicHiba.jsp
- index.jsp

A felület egységes megjelenítéséhez a CSS programozási nyelv nyújtotta lehetőségeket használtam ki. A könnyebb kezelhetőség, és a rendezettség miatt felosztottam a felületet egy #lap-ra, ami 1200px széles, és ezt daraboltam tovább #fejléc, #közép, és #lent azonosítójú egységekre. A #fejléc, és a #lent minden oldalon ugyanúgy jelenik meg, a fejléc tartalmazza a Hallgatói fórum főcímet, a #lent pedig a Készítette: Hóbel Tímea paragrafust. A #közep kapja a legtöbb hangsúlyt, ide kerül mindig az oldal aktualitása, ezért a JSP-k leírásakor csak ezt az egységet részletezem.

A használt CSS forráskód a következő oldalon lévő ábrán tekinthető meg.

```
<style type="text/css">
    #lap{width:1200px;
         margin:10px auto 10px auto;}
    #fejlec, #kozep, #lent{margin:0 0 0 0;}
    #fejlec{height:150px;
            letter-spacing: 5px;
            font: italic bold;
            color: white;}
    #fejlec>h1{ font-size: 32px;
                text-align: center;
                text-shadow: 8px 8px 8px black;}
    #kozep{ height: 480px;
            color: white;}
    #kozep>h2{font-size: 22px;
              text-align: left;
              text-shadow: 5px 5px 5px black;}
    #lent{text-shadow: 3px 3px 3px black;
          text-align: center;
          font: italic bold;
          font-size: 12px; color: white;
          padding:10px 0 10px 0;
          height: 12px;}
    body{background-color: grey;
         font-family: Comic, sans;}
</style>
```

A CSS forráskód

AddNewFelhasznalo.jsp

A regisztrációhoz tartozó felületet definiálja. A főoldalról a Regisztráció gomb megnyomásával juthatunk el erre az oldalra. Új felhasználó regisztrálásához ki kell tölteni minimum a táblázat név, és jelszó mezőjét, a születési év opcionális. Amennyiben a kötelező mezők valamelyike hiányzik vagy egy más létező felhasználónévvel próbálunk meg újra regisztrálni, akkor hibát kapunk, és a RegHiba.jsp által definiált oldalra jutunk. Sikeres regisztrációról akkor beszélünk, ha érvényes, és még nem foglalt nevet, illetve jelszót adtunk meg, és a Regisztráció mentése gombra kattintva visszajutunk a bejelentkezés oldalára. Ha mégsem szeretnénk regisztrálni, akkor lehetőségünk van a Vissza a főoldalra gombbal a bejelentkezési oldalra navigálni.

ListazFelhasznalok.jsp

A Témák című oldalról a Taglista lekérdezése linkre kattintva jutunk el erre a felületre, ahol a fórumra regisztrált felhasználók listáját olvashatjuk. Egyetlen irányba tudunk továbblépni, Vissza a témákhoz gombbal vissza az előző oldalra.

ListazHozzaszolasok.jsp

Erre a felületre akkor kerülünk, ha a Témák című oldalon lévő Témák felirat alatti felsorolásból választunk egy nevet, és rákattintunk. Ezzel eljutunk a témához tartozó hozzászólások oldalára, ahol az eddigi hozzászólásokat olvashatjuk, és fűzhetünk további megjegyzéseket a témához. Új hozzászólás hozzáfűzéséhez az input mezőbe kell begépelnünk a megjegyzésünket majd a Megosztás gombra kell kattintanunk. Egy irányba léphetünk tovább, vissza az előző oldalra a Vissza a témákhoz gombbal.

ListazTopics.jsp

Ezzel a felülettel találkozunk, ha sikeresen bejelentkeztünk a fórumra. A Témák felirat alatt olvashatók a fórumon eddig létrehozott témák neveit. Közvetlenül ez alatt van lehetőségünk új téma létrehozásához. Az input mezőbe begépeljük a létrehozandó téma nevét, majd a Létrehozás gombra kattintunk. Kitöltetlen mező esetén nem hozható létre téma, ha mégis előfordulna, hogy üres input mező esetén kattintunk a Létrehozás gombra, akkor a TopicHiba.jsp által definiált oldalra jutunk, ahol figyelmeztet minket az alkalmazás, hogy hibásan töltöttük ki a mezőt. Ezalatt található még egy link, amivel a fórumra regisztrált felhasználók listája tekinthető meg, ez a Taglista lekérdezése hivatkozás. Amennyiben el szeretnénk hagyni a fórumot, akkor a Kijelentkezés gombra kattintva visszajutunk a főoldalra.

LoginHiba.jsp

Csak a bejelentkezés folyamán elkövetett hibák következtében juthatunk el erre az oldalra. Ilyen hiba lehet, ha elrontjuk a felhasználónevet vagy a jelszót, illetve ha az egyik mezőt üresen hagyjuk.

RegHiba.jsp

A regisztráció során bekövetkezett hibák hatására kerülünk erre az oldalra. Ilyen hiba adódhat, ha a regisztráció során már meglévő felhasználónevet próbálunk megadni, illetve ha valamelyik kötelezően kitöltendő mezőt (név, és jelszó) üresen hagyjuk.

TopicHiba.jsp

Akkor kerülünk erre a felületre, ha a Témák című oldalon üresen hagyjuk a téma létrehozásakor kitöltendő input mezőt.

index.jsp

Az alkalmazás kezdőoldala. A belépés felülete.

7.4 A hallgatói fórum tesztelési terve

Ezt a dokumentumot megtekinthetjük az 2. számú függelékben.

7.5 Teszt specifikáció

Ezt a dokumentumot megtekinthetjük az 3. számú függelékben.

7.6 Maga a tesztelés

A tényleges tesztelés megkezdése előtt fontos kiválasztani a megfelelő teszteszközt, és kialakítani a környezetet. A választásom egy korábbi fejezetben bemutatott Selenium teszteszközre esett. A tesztelést a Selenium IDE használatán keresztül mutatom be. Ezzel az eszközzel elsősorban tesztrobotok készítését végezzük. A felületén elhelyezett Rec gomb funkciója rendkívül meggyorsítja ezt a lépést. A gomb bekapcsolásával az IDE rögzíti a felületen végrehajtott eseményeket, így nem kell nekünk egyesével kikeresnünk a linkek címeit vagy a gombok azonosítóit stb. A tesztelni kívánt webes alkalmazást megnyitjuk egy Firefox böngészőben, ehhez a Forum projektet be kell importálni Netbeansbe. A futtatáshoz szükséges telepíteni egy Apache Tomcat 6.0 szervert. Ha nem a Firefox az alapértelmezett böngészőnk, akkor a futtatás hatására megnyílt böngésző címsorából másoljuk át a Firefox böngésző címsorába a Hallgatói fórum címét. Nincs más teendőnk, mint az Eszközök menüpont alatt található, már korábban installált Selenium IDE eszközt kiválasztjuk, és megnyitjuk valamelyik szkriptet. A felület néhány fontos elemét már korábban bemutattam, most a konkrét feladat teszt szkriptjének megírását mutatom be. A tesztesetek mindig egy open(url) paranccsal kezdődnek, ezzel azonosítva az ellenőrizendő oldal címét. Ezután következhetnek az alkalmazás manipulálására szolgáló parancsok. A fórum esetén tesztelni fogjuk az alkalmazás használhatóságát, működését, az alapfunkcióit. A megírt szkriptek közül

a smoke_test tesztesetet fogom lépésenként bemutatni, a többi szkript esetében csak egy működési leírás készül.

Command	Target	Value
open	http://localhost:8080/index.jsp	
type	nev	admin
type	jelszo	admin
click	login	
waitForPageToLoad	30000	
assertTitle	Témák	
click	link=Statisztika	
waitForPageToLoad	30000	
assertTitle	Hozzászólások	
waitForTextPresent	A topic neve	
type	text	új hozzászólás
clickAndWait	newShareButton	
click	backToTopicButton	
waitForPageToLoad	30000	
assertTitle	Témák	
click	link=Statisztika	
waitForPageToLoad	30000	
assertTextPresent	új hozzászólás	
echo	Létezik az "új hozzászólás szövege" hozzászólás.	
click	backToTopicButton	
waitForPageToLoad	30000	
assertTextPresent	Témák	
click	listUsersButton	
waitForPageToLoad	30000	
assertTitle	Felhasználók	
waitForTextPresent	Regisztrált felhasználók listája:	
click	backToTopicButton	
waitForPageToLoad	30000	
assertTitle	Témák	
clickAndWait	logoutButton	
waitForPageToLoad	30000	
assertTitle	Login	

Az IDE smoke_test felülete

7.6.1 A smoke_test szkript bemutatása

Egy alkalmas smoke teszt lehet, ha a fórumra belépünk egy regisztrált felhasználóval, egy meglévő témához hozzászólunk, és leellenőrizzük, hogy ez a hozzászólás megjelenik-e a téma hozzászólásai között. Majd kilépünk. Ezeknek a lépéseknek, és funkcióknak az ellenőrzése a

szoftver evolúciója során is fontos lesz, ezért célszerű lehet egy tesztrobot megírása. Ehhez a tesztrobothoz tartozó szkript lépései:

- open(http://localhost:8080/index.jsp/): az open paranccsal a tesztelendő oldal URL címét adhatjuk meg, azaz megnyitjuk a fórum kezdőoldalát;
- type(nev, admin): az első paraméterben megadott nev ID-jű input mezőbe beírja a második paraméterben megadott ”admin” sztringet, azaz begépeljük a felhasználónevet;
- type(jelszo, admin): az első paraméterben megadott jelszo ID-val rendelkező input mezőbe begépeli a második paraméterben megadott ”admin” sztringet, azaz begépeljük a jelszót;
- click(login): a login azonosítóval rendelkező felületi elemre kattint, azaz a Belépés gombra kattintunk;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Témák című oldal betöltődésére;
- assertTitle(Témák): ellenőrzést végez, a paraméterben megadott ”Témák” sztringet hasonlítja az adott oldal címével, ha eltérést tapasztal, akkor hibával leáll a futás;
- click(link=Statisztika): a paraméterben megadott Statisztika nevű linkre kattint, azaz a Statisztika nevű témára kattintunk;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Hozzászólások című oldal betöltődésére;
- assertTitle(Hozzászólások): ellenőrzést végez, a paraméterben megadott Hozzászólások sztringet hasonlítja az adott oldal címével, ha eltérést tapasztal, akkor hibával leáll a futás;
- waitForTextPresent(A topic neve): a paraméterben megadott ”A topic neve” sztringet keresi a felületen, ha egy megadott időn belül nem találja, akkor hibával leáll a futás;
- type(text, új hozzászólás): az első paraméterben megadott text ID-val rendelkező input mezőbe begépeli a második paraméterben megadott ”új hozzászólás” sztringet, azaz beírunk egy új hozzászólást;
- clickAndWait(newShareButton): a newShareButton azonosítóval rendelkező felületi elemre kattint, és megadott ideig vár, mire az oldal újra betöltődik, azaz a Megosztás gombbal közzétesszük a hozzászólást;

- click(backToTopicButton): a backToTopicButton azonosítóval rendelkező felületi elemre kattint, azaz visszairányít a Témák oldalra;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Témák című oldal betöltődésére;
- assertTitle(Témák): ellenőrzést végez, a paraméterben megadott ”Témák” sztringet hasonlítja az adott oldal címével, ha eltérést tapasztal, akkor hibával leáll a futás;
- click(link=Statisztika): a paraméterben megadott Statisztika nevű linkre kattint, azaz a Statisztika nevű témára kattintunk;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Hozzászólások című oldal betöltődésére;
- assertTextPresent(új hozzászólás): a paraméterben megadott ”új hozzászólás” sztringet keresi a felületen, amennyiben a keresés eredménytelen, akkor a futás hibával leáll. Ha sikerrel jár a keresés folyamán, akkor a következő parancs hatására kiírjuk, hogy létezik a hozzászólás;
- echo(Létezik az "új hozzászólás szövege" hozzászólás.): a paraméterben megadott "Létezik az "új hozzászólás szövege" hozzászólás. " sztringet kiírja a Log fülre
- click(backToTopicButton): a backToTopicButton azonosítóval rendelkező felületi elemre kattint, azaz visszairányít a Témák oldalra;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Hozzászólások című oldal betöltődésére;
- assertTextPresent(Témák): a paraméterben megadott ”Témák” sztringet keresi a felületen, amennyiben a keresés eredménytelen, akkor a futás hibával leáll;
- click(listUsersButton): a listUsersButton azonosítóval rendelkező felületi elemre kattint, azaz a felhasználók ki listázásának felületére irányít;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Témák című oldal betöltődésére;
- assertTitle(Felhasználók): ellenőrzést végez, a paraméterben megadott ”Felhasználók” sztringet hasonlítja az adott oldal címével, ha eltérést tapasztal, akkor hibával leáll a futás;
- waitForTextPresent(Regisztrált felhasználók listája:): a paraméterben megadott ”Regisztrált felhasználók listája:” sztringet keresi a felületen, ha egy megadott időn belül nem találja, akkor hibával leáll a futás;

- click(backToTopicButton): a backToTopicButton azonosítóval rendelkező felületi elemre kattint, azaz visszairányít a Témák oldalra;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Hozzászólások című oldal betöltődésére;
- assertTitle(Témák): ellenőrzést végez, a paraméterben megadott ”Témák” sztringet hasonlítja az adott oldal címével, ha eltérést tapasztal, akkor hibával leáll a futás;
- clickAndWait(logoutButton): a logoutButton azonosítóval rendelkező felületi elemre kattint, és megadott ideig vár, mire az oldal újra betöltődik, azaz a kilépünk a fórumból;
- waitForPageToLoad(30000): a paraméterben megadott ezredmásodpercnyi ideig várunk az új oldal betöltődésére, azaz a Hozzászólások című oldal betöltődésére;
- assertTitle(Login): ellenőrzést végez, a paraméterben megadott ”Login” sztringet hasonlítja az adott oldal címével, ha eltérést tapasztal, akkor hibával leáll a futás.

7.6.2 A smoke_reg_test szkript bemutatása

Hasznos lehet egy olyan smoke teszt automatizálása, amely a belépés előtt regisztrál egy új felhasználót, és utána ellenőrizni a többi funkciót. Ez a tesztrobot is a fórum kezdőoldalának megnyitásával indul, majd egyből a regisztrációs oldalra irányít át, ahol berögzít egy ”név” felhasználónevű hallgatót. Az új felhasználónévvel belép a rendszerbe, létrehoz egy ”Szakdolgozat” nevű témát, majd ellenőrzi, hogy az ”1. hozzászóló” sztring létezik-e a felületen, ezzel vizsgálva azt is, hogy automatikusan legenerálódott-e az első hozzászólás a téma létrehozásakor. Ezután kilép a rendszerből.

7.6.3 A login_test szkript bemutatása

Ez a tesztrobot kifejezetten a belépési követelmények ellenőrzésére szolgál. Megnyitja a fórum kezdőoldalát, ami a belépési oldal, és itt csak a felhasználónév attribútumhoz tartozó input mezőt tölti ki, a jelszó input mezőt üresen hagyja, és vizsgálja, hogy jelez-e a rendszer a hibáról. Ugyanezt a vizsgálatot fordított mező kitöltéssel is elvégzi, azaz a felhasználónévhez tartozó input mezőt hagyja üresen, miközben a jelszó mezőt kitölti. Ezután a két előre berögzített felhasználó létezését ellenőrzi, belép mind az admin, mind pedig a user felhasználóval, és ha mindkét esetben sikerült belépnie a fórumra, akkor visszalép, és az egyik

esetben elrontja a jelszót, ezzel azt vizsgálva, hogy, észreveszi-e a rendszer, hogy egy felhasználónévhez hibás jelszót gépeltünk be. Végül visszalép a kezdőoldalra.

7.6.4 A reg_test szkript bemutatása

Ez a tesztrobot a regisztráció ellenőrzését végzi. Kezdésként megnyitja a fórum nyitó oldalát, majd a Regisztráció gombbal átirányít a regisztrációs oldalra. Annak az esetnek a vizsgálatával kezd, amikor csak a felhasználónév attribútumhoz tartozó input mezőt töltjük ki. Amennyiben a rendszer észleli a hibát, akkor a következő eset a jelszó attribútumhoz tartozó input mező hiányos kitöltésének vizsgálata. A helyes működés ebben az esetben is az, ha a rendszer jelzi, hogy helytelenül próbáltunk meg regisztrálni. Ezután következik az az eset, amikor már korábban regisztrált felhasználónévvel próbálunk meg regisztrálni. Megvizsgálja, hogy tudunk-e admin, illetve user felhasználónévvel regisztrálni. Ekkor is jelez a rendszer, hogy helytelen a regisztrációnk. Végül egy helyes felhasználónévvel kitölti a névhez tartozó input mezőt, és begépel egy jelszót is. Amennyiben ezzel a felhasználónévvel sikerül belépnie a fórumra, akkor sikeresen lefutott a teszt sztkript és kilép a rendszerből.

7.6.5 A topic_test szkript bemutatása

Kifejezetten a témákkal kapcsolatos követelményeket ellenőrzi. A tesztrobot megnyitja a kezdőoldalt, és egy már korábban regisztrált felhasználóval belép. Leellenőrzi a két, már előre berögzített téma a ”Statisztika”, és a ”Mesterséges intelligencia” létezését, majd az új téma létrehozására vonatkozó megszorításokat vizsgálja. Megpróbál üres input mező esetén létrehozni témát, illetve már létező téma nevet megadni. Mindkét esetben a rendszernek fel kell ismernie a helyzeteket, és jelezni a hibát. Ezután létrehoz egy ”Szakdolgozat” nevű témát. Amennyiben legenerálódott hozzá az automatikus első hozzászólás, akkor kilép a rendszerből.

7.6.6 A share_test szkript bemutatása

A témákhoz kötődő hozzászólásokat vizsgálja. Egy létező felhasználónévvel belép a rendszerbe, és kiválasztja a ”Statisztika” témát. Vizsgálja a témához tartozó első hozzászólást. Ezután kiválasztja a ”Mesterséges intelligencia” témát, és itt is elvégzi az automatikusan legenerálódott első hozzászólás létezését. Ezután megoszt egy tetszőleges sztringet tartalmazó hozzászólást. Végül kilép a rendszerből.

7.6.7 A loginList_test szkript bemutatása

Ez a tesztrobot csak annak felhasználói statisztikának az ellenőrzését végzi, amelyik a felületre kilistázza a regisztrált felhasználókat. Regisztrál egy ”loginName” nevű felhasználót, amivel belép a rendszerbe, és a Témák oldalról a ”Taglista lekérdezése” linkkel átirányít a felhasználókat megjelenítő felületre. Végül kilép a rendszerből.

7.6.8 A functional_test szkript bemutatása

Ez a tesztrobot összefoglalva vizsgálja a rendszer működését. Minden nagyobb funkciót érint, mint a regisztrációt, belépést, témalétrehozást, hozzászólás megosztást, és a taglista lekérdezést.

7.7 Tesztjelentés

Ezt a dokumentumot megtekinthetjük a 4. számú függelékben.

8. Összegzés

A szakdolgozatom írása során célom volt, hogy mélyebb betekintést nyújtsak az alkalmazások tesztelésének folyamatába. Nyilvánvalóvá vált számomra, hogy ez a szakma is egyre inkább előtérbe kerül. Rengeteg eszköz közül nyílik lehetőségünk választani, az én választásom a Seleniumra esett, mivel ezzel az eszközzel csökkenthető a tesztelésre szánt idő, és ez által a költség is. Ezek mellett természetesen szükség van emberi erőforrásra, mivel egy program sem képes teljesen átlátni az összefüggéseket, illetve a jól működő robotokat is meg kell írni. A Selenium használata egyre szélesebb körben kezd elterjedni, amiben több tényező is szerepet játszik. Az egyik, hogy ingyenesen használható a szoftver, a másik pedig, hogy egyszerűen, és gyorsan elsajátítható a használata. Az automatizált teszteléshez implementáltam egy mini webes alkalmazást, a Hallgatói fórumot. Majd a valóságnak megfelelően mutattam be egy szoftvertermék születése során zajló tesztelési fázist.

9. Irodalomjegyzék

1. Virginia DeBolt (2005): HTML és CSS – Webszerkesztés stílusosan, Kiskapu kiadó

2. Dorothy Graham, Erik Van Veenendaal, Isabel Evans, Rex Black (2010): A szoftvertesztelés alapjai, Alvicom Kft. kiadó
3. Selenium http://seleniumhq.org/
4. Netbeans http://netbeans.org/
5. Apache Tomcat http://tomcat.apache.org/
6. CSS http://www.w3.org/
7. CSS http://weblabor.hu/cikkek/designbolkeszoldal
8. Servlet, JSP tutorial http://www.servletworld.com/
9. Tesztelés http://teszteles.blog.hu/
10. Tesztelés http://www.dcs.vein.hu/~simon/oktatas/ST/23.pdf
11. Tesztelés http://hu.wikipedia.org/wiki/Szoftvertesztel%C3%A9s
12. Tesztelés http://www.cid.com.ro/ksimon/svv/vv_kurzus3.pdf

10.Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani:

- Dr. Kósa Márk Szabolcs Tanár Úrnak, amiért elvállalta a szakdolgozatom témavezetését. Valamint köszönöm az egész munka alatt tanúsított rugalmas hozzáállását, és a rendkívül inspiráló tanácsait.
- Boda Bélának a szakdolgozatom témaválasztásában nyújtott segítségért valamint a szakmai segítségért, amit az elmúlt hónapok során nyújtott.
- Antal Orsolyának, és Farkas Évának a folyamatos támogatásért mind a dolgozat írása, mind pedig az egyetemi éveim során.

11. Függelék

1. számú függelék

A hallgatói fórum követelményspecifikációja

Követelményspecifikáció.docx

2. számú függelék

A követelményspecifikáció alapján elkészített tesztelési terv

Tesztelési terv.docx

3. számú függelék

A tesztelési terv alapján elkészített teszt specifikáció

Teszt specifikáció.docx

4. számú függelék

A teszt specifikációban leírt tesztesetek végrehajtásáról szóló tesztjelentés

Tesztjelentés.docx