Debreceni Egyetem Informatikai Kar

Webes alkalmazásfejlesztés szabadon választott témakörben

Témavezető:
Dr. Kuki Attila
Egyetemi adjunktus

Készítette:
Uzonyi Éva Nikoletta
Programtervező matematikus

Debrecen,
2009

Tartalomjegyzék

1. Bevezetés

Napjainkban szinte minden a számítógép körül forog. Az élet minden területén előfordul, mindenfelé használják. Számítógépet alkalmaznak például a gépgyártásban, az egészségügyben, a bankokban, a hipermarketekben, csakhogy néhány területet említsek. Azonban ne feledkezzünk meg arról, hogy az ember a mindennapjait ma már szinte el sem tudná képzelni számítógép nélkül. Mikor az első elektronikus számítógép, az ENIAC 1946-ban elkészült, még biztosan nem gondolták, hogy egy napon ennyire uralkodóvá válik. Manapság nem szükséges elmenni a közeli boltba megvásárolni egy könyvet, ruhadarabot vagy valamilyen ajándékot, elég leülni a számítógépünk elé nyáron a hűvös szobában vagy télen a kandalló mellé a fotelbe, és máris a webes áruházak széles tárháza tárul elénk. Nem csak a különböző cégek, egyesületek, hanem egyre gyakrabban magánszemélyek is készítenek saját weboldalt.

Az Internet egyre növekvő népszerűségének köszönhetően nem elegendőek az állandó tartalommal rendelkező oldalak, napjainkban már szinte csak a dinamikusan változó, nagy információtömeget hordozó oldalak számíthatnak nagy látogatottságra. Ezt a nagyméretű információigényt, valamint az interaktivitást nem lehetséges egyszerű HTML oldalakkal megvalósítani, szükség van az adatok tárolásához egy adatbázisrendszerre, valamint az adatbázis adatait feldolgozó webes alkalmazásra is.

2. Témaválasztás és célkitűzés

Nagyon sokat gondolkoztam azon, hogy hogyan tudnám egy diplomamunka keretein belül átadni azt a tudásmennyiséget, amit az elmúlt öt év során sikerült elsajátítanom. Milyen témát válasszak, aminek a segítségével be tudnám mutatni mind az elméleti, mind pedig a gyakorlati tudásomat? Számításba vettem például egy online könyvtár megalkotását vagy egy webáruház elkészítését. Azonban nem akartam azt, hogy egy tömegesen jelen lévő dolgot készítsek, valami egyedit szerettem volna megalkotni. Ekkor jött az ötlet családi hagyomány alapján, a veterán jármű kölcsönzése.

Nagyszülőkig visszamenőleg megfigyelhető családomban a járművek, és elsősorban a motorok szeretete. Ezt a „vérvonalat” viszem én is tovább. Rajongok az autókért és motorokért, és vezetem is őket. Eljátszottam a gondolattal, hogy ha Interneten keresztül is hozzá lehetne jutni egy-egy darabhoz, akkor én mit választanék. Tagja vagyok a hajdúböszörményi Veterán Kulturális és Sport Egyesületnek, innen jött az ötlet, hogy valami veterán csodát választanék, valami kuriózumot, még ha néhány órára is. Ezek a járművek szépek ugyan, de az áruk is borsos, így nem engedheti meg magának bárki, hogy egy ilyet birtokoljon, de vannak olyan jeles napok, alkalmak, amikor pár órára vagy esetleg néhány napra megéri áldozni rá. Ehhez adhat segítséget egy Veterán Autó- és Motorkölcsönző.

Ezzel elérkeztünk a diplomamunkám témájához, a webes alkalmazásfejlesztéshez. Segítségével be szeretném mutatni a ma oly elterjedt webes alkalmazások teljes fejlesztési életciklusát: az ötlet megszületésétől, a tervek megalkotásán és megvalósításán át, az alkalmazás elkészülte után következő feladatokig. Egyben szeretném bemutatni az egyetemen megszerzett elméleti tudásomat egy webes alkalmazás megalkotásán keresztül. Az alkalmazás tehát egy online Veterán Autó- és Motorkölcsönző rendszer, mely tökéletes példája a mai webes alkalmazások működésének, a dinamikusan változó tartalomnak, valamint az egyes felhasználói igények teljesítésének.

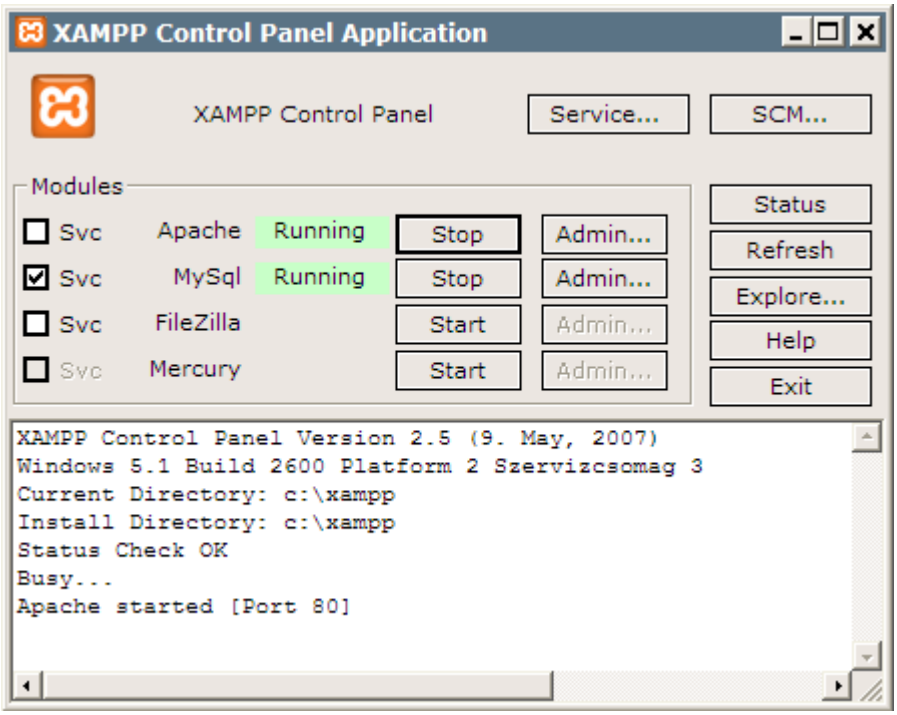
3. Fejlesztői környezet

Manapság egy-egy alkalmazás megvalósításához több lehetőség áll rendelkezésünkre.

3.1. XAMPP

A XAMPP fejlesztői környezet alkalmas mind Windows, mind Linux, illetve Solaris rendszereken történő fejlesztésekhez. Magában foglalja az Apache webkiszolgálóz, a MySQL adatbázisszervert, a PHP és Perl fordítót, valamint a phpMyAdmin felületet.

Egy ingyenes programcsomag, melyet bárki letölthet az Internetről, és több telepítési mód közül választva egyszerűen installálhatja számítógépére. A fejlesztői környezet rendkívül egyszerűen kezelhető a Control Panel segítségével.



3.2. Apache

Az Apache szabad, az Interneten a legszélesebb körben használt web-szerver. Az Illinois-i egyetemen készítették ez az 1990-es évek közepén. A megalkotók célja egy olyan web-szerver megalkotása, karbantartása volt, amely megfelel a gyorsan változó Internet követelményeinek, emellett biztonságos, üzleti és vállalat felhasználásra is megfelelő, és szabadon használható.

3.3. MySQL

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver. Az egyik legelterjedtebb adatbázis-kezelő, aminek okai lehetnek:

- saját számítógépen történő telepítés és használat céljából a program ingyenesen letölthető
- egyszerűen telepíthető több operációs rendszeren
- egyszerűen kezelhető
- gyors és hatékony kliens-szerver rendszert valósít meg, mely rendkívül nagy és összetett adatbázisokkal is könnyedén megbírkózik, és ezáltal nagyobb projektekben is kiválóan alkalmazható

3.4. PHP

A PHP alkalmazásával dinamikus weboldalakat hozhatunk létre. A dinamikus tartalommal rendelkező oldalak tartalma és struktúrája akár minden elérésnél más lehet. Manapság szinte minden weboldal esetén szükség van a dinamikus tartalomra.

A PHP-t Rasmus Lerdorf 1995-ben saját, Interneten közzétett önéletrajzának elérhetőségének nyomonkövetése végett alkotta meg. Ez egyszerű makrónak szánt program volt, ami idővel túlnőtt a kezdeti célon, és egy teljes funkcionalitással rendelkező nyelvvé vált.

A PHP könnyen beágyazható a HTML nyelvű oldalakba, és kiválóan képes kommunikálni a hálózat adatkommunikációs protokolljával, a http-el.

A PHP programok jól hordozhatóak, és biztonságosak. Együttműködnek a különböző szerverekkel, operációsrendszerekkel és adatbázis kezelőkkel.

A PHP szerveroldali beágyazott nyelv, azaz a PHP programok a szerveren olyan módon hajtódnak végre, hogy a PHP interpreter a megfelelő állományokat átfésüli, értelmezi, majd a végrehajtás után HTML oldalakba illeszti a kimenetet. Ennek előnyei a következők:

- a szerveroldali szkriptek fejlesztése teljesen független a felhasználó által alkalmazott böngészőtől
- a szerveroldali szkriptek forrásnyelvi változata nem olvasható a felhasználó böngészőjében

– a HTML oldalak mérete csökken, így maga a weboldal beolvasása is gyorsabbá válik

3.5. phpMyAdmin

A phpMyAdmin egy nyílt forrású eszköz, amit PHP-ban írtak a MySQL menedzselésére az Interneten keresztül. Használatával lehetőségünk nyílik adatbázisok létrehozására, eldobására, táblák módosításáram törlésére, módosításáram bővítéséra, valamint SLQ parancsok futtatására. Nagy kényelmet jelent az importálás és az exportálás lehetősége.

4. A szoftverfejlesztés életciklusa

A szoftverkrízis oda vezetett, hogy az 1960-as évek végére felismerték, hogy a szoftver termék, és mint termék az előállítása egy folyamat, mely különböző fázisokból áll. Minden fázis tevékenységsorozat, és minden tevékenység végén előáll valamilyen eredmény, részeredmény. A szoftverfejlesztésben alapvetően négy nagy tevékenységsorozatot különböztetünk meg:

- specifikáció: az elvárások feltárása, konkretizálása, rögzítése
- elkészítés/implementáció
- validáció: az elkészült szoftverről be kell látni, hogy megfelelő
- szoftverevolúció: az elkészült szoftver módosítása, továbbfejlesztése (nem hibajavítás!)

A fejlesztés jelentős része gondolati síkon folyik. Modellekben gondolkodunk, vagyis a valóságos dolgokról a magunkban kialakult képeket jelenítjük meg a modellekben. Egy modell éppen ezért sohasem tükrözheti pontosan a valóságot, mivel a modellben szükségképpen azokra a tulajdonságokra helyezzük a hangsúlyt, melyek céljaink elérése szempontjából nélkülözhetetlenek. Ezért a modell szükségképpen absztrakt. A szoftverfejlesztési életciklusban tehát egy absztrakt megközelítéstől egy konkrét megközelítés felé haladunk valamilyen alkalmazott tevékenységsorozat folyamán. Az elmúlt negyven év arról szólt, hogy hogyan lehetne eltolni ezt a tevékenységsorozatot abba az irányba, hogy

minél hamarabb, minél absztraktabb módon megfogalmazott tulajdonságokból tudjuk a konkrét terméket automatikusan létrehozni.

A folyamat:

- követelmények meghatározása és elemzése
- tervezés
- alrendszerek fejlesztése
- rendszerintegráció
- a rendszer telepítése
- rendszer működtetése
- rendszerevolúció
- a rendszer üzemen kívül helyezése.

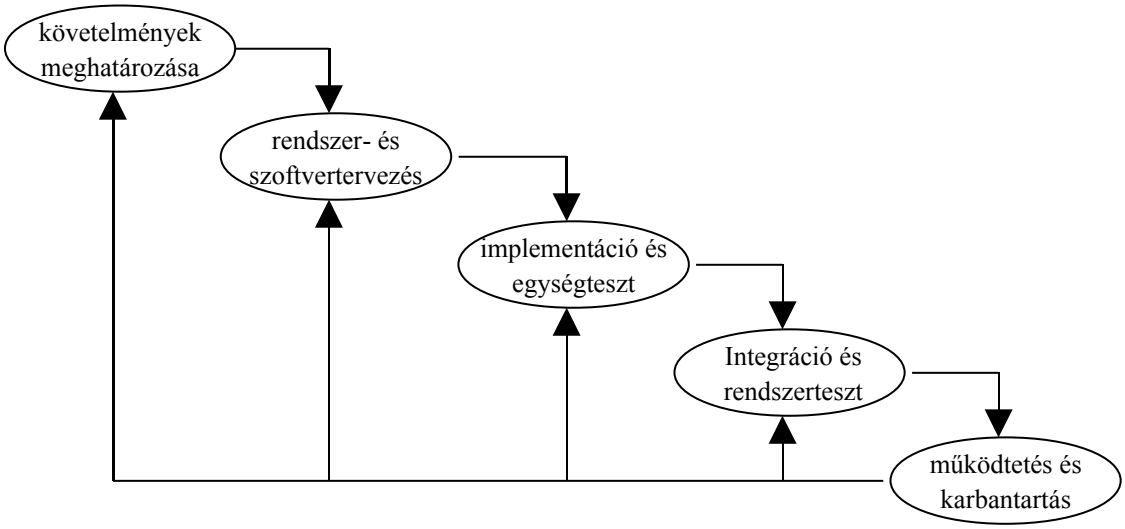
A fejlesztés bonyolult folyamat, így a szoftverfolyamat is modellezhető, amely nem más, mint a folyamat egy absztrakt reprezentációja. Minden egyes folyamatmodell különböző speciális nézőpontból reprezentál egy folyamatot, mindegyiknek megvan a maga előnye és hátránya, valamint az alkalmazási területe, vagyis, hogy mikor célszerű használni, illetve mikor nem. A folyamatmodellek az elmúlt harminc éven át folyamatosan alakultak ki, azonban ez nem jelenti azt, hogy a korábbiak elavultak lennének. A dolgozatomban szereplő folyamatmodellek mindegyike ma is él, azonban el kell dönteni, hogy egy adott szoftver fejlesztésénél melyik modellt követjük, természetesen beleértve azt is, hogy az egyes modelleket keverhetjük: egy adott lépésben, az adott részfolyamatban az oda legjobban megfelelő folyamatmodell alkalmazandó. Ezzel egyes előnyök kiemelhetők, ezzel együtt csökkenthetők a modell alkalmazásával járó esetleges hátrányok.

4.1. Vízesésmodell

A szoftverfejlesztés problémáinak felismerése után a projekt folyamatának leírására kialakult legrégebbi modell az 1970-es években kialakult vízesésmodell**,** amelyet fázismodellnek is nevezünk. A modell a termékfejlesztésre koncentrál, azaz az első működő példány előállításáig terjedő szakasz lefolyását ábrázolja. Ma általában a változtatásokat is

kezelő ciklikus modellekbe ágyazva használják, önmagában legfeljebb nagy egyedi szoftverek esetén fordul elő.

Nevét a szemléltető ábra jellegzetes alakjáról kapta: a folyamatelemek, az egyes lépések meglehetősen mereven egymásra épülnek, lépcsőzetesen kapcsolódnak egymáshoz, mint ahogy a víz folyik alá a sziklákon.



A modell alapvető szakaszai az alábbi fejlesztési tevékenységekre képezhetők le:

1. Követelmények meghatározása: a rendszer szolgáltatásai, megszorításai és céljai a rendszer felhasználóival történő konzultáció alapján alakulnak ki, ezeket később részletesen kifejtik, és ezek szolgáltatják a rendszer-specifikációt.
2. Rendszer- és szoftvertervezés: a rendszer tervezési folyamatában válik ketté a hardver- és a szoftverkövetelmény. Itt kell kialakítani a rendszer átfogó architektúráját. A szoftver tervezése alapvető szoftverrendszer-absztrakciók, illetve a közöttük lévő kapcsolatok azonosítását és leírását is magában foglalja.
3. Implementáció és egységteszt: ebben a szakaszban a szoftverterv programok, illetve a programegységek halmazaként realizálódik. Az egységteszt azt ellenőrzi, hogy minden egyes egység megfelel-e a specifikációjának.
4. Integráció és rendszerteszt: megtörténik a különálló programegységek, illetve programok integrálása és teljes rendszerként történő tesztelése, hogy fény derüljön

arra, hogy a rendszer megfelel-e a követelményeknek, a tesztelés után a szoftverrendszer átadható.

5. Működtetés és karbantartás: általában ez a szoftver életciklusának leghosszabb fázisa. Megtörtént a telepítés és a rendszer gyakorlati használatbavétele. A karbantartásba beletartozik az olyan hibák kijavítása, amelyekre nem derült fény az életciklus korábbi szakaszaiban, a rendszeregységek implementációjának továbbfejlesztése, valamint a rendszer szolgáltatásainak továbbfejlesztése a felmerülő új követelményeknek megfelelően.

A lépcsők fázisokat jelölnek. Egy fázis lezárulta után lehet hozzákezdeni a következő fázishoz. A fázisok lineárisan következnek egymás után. Ez a modell előnyeit, és hátrányait egyaránt magában foglalja.

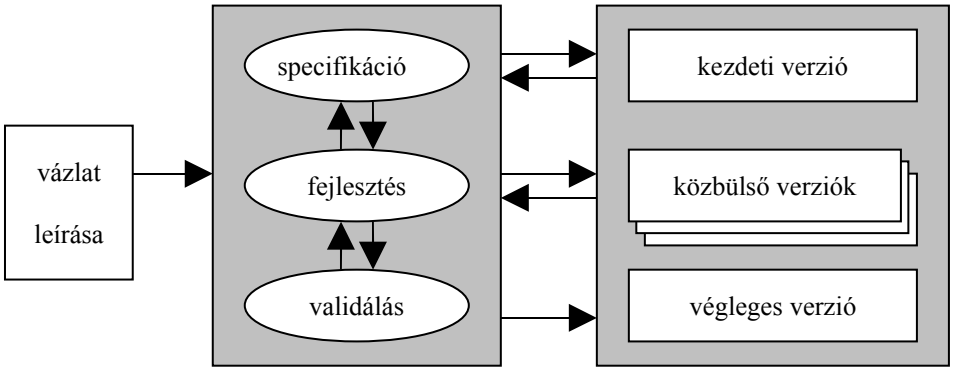
A vízesésmodell nagyon jól menedzselhető, szervezhető, tervezhető. Azonban egy nagyon merev modellről van szó, nem flexibilis, nagyfokú az egymásra épülés. Ez a merevség oda vezet, hogy egy implementációs tervnek teljes egészében készen kell lennie. Minden egyes lépés végén születik egy dokumentum: az első és második lépés esetén tényleges szöveges, rajzos dokumentum, a harmadik lépés után egy szoftver a hozzá tartozó dokumentumokkal. Ha ezt a dokumentumot a felhasználó és az informatikus együtt elfogadja, akkor lehet rátérni a negyedik lépésre. Előnyei a kis- és közepes méretű rendszereknél, valamint azoknál a rendszereknél jelentkezik, melyeknél a követelmények jól specifikáltak. Ekkor egy jól strukturált, jól felépített, jó architektúrális jellemzőkkel rendelkező, robosztus rendszer fejleszthető.

Ez is, mint minden elvi modell, absztrakt, azonban a gyakorlatban az egyes lépések átfedik egymást, de mindenképpen igaz, hogy a modellben nincs párhuzamos fejlesztés, a lépések semmiképpen sem párhuzamosak.

4.2. Evolúciós modell:

Az 1980-as években alakult meg a vízesésmodell tagadására, nagy rendszerek megalkotására hivatott. Egyik alapötlete, hogy a vízesésmodellnél oly jellemző abszolút

linearitással szemben próbáljunk meg egy párhuzamos fejlesztési modellt kialakítani, azaz párhuzamosítsuk az egyes lépéseket. A másik ötlet, hogy a fejlesztést úgy végezzük el, hogy egy nagyon korai implementációt készítünk, majd az implementációk verzióin keresztül jutunk el a végső verzióhoz. Tehát az evolúciós fejlesztés verziósorozatot produkál.



A modell lényege tehát, hogy rendelkezésünkre áll egy vázlatos követelményrendszer, és a követelményspecifikáció, az implementálás és validálás párhuzamosan úgy történik, hogy adott egy kezdeti implementáció, majd kialakulnak különböző közbülső változatai, míg végül megszületik a végleges verzió. Ezeket a verziókat hívja a modell prototípusoknak. A párhuzamosság és a verziókezelés nagyon gyors visszacsatolási lehetőséget jelent.

A prototípusok két fajtáját különböztetjük meg:

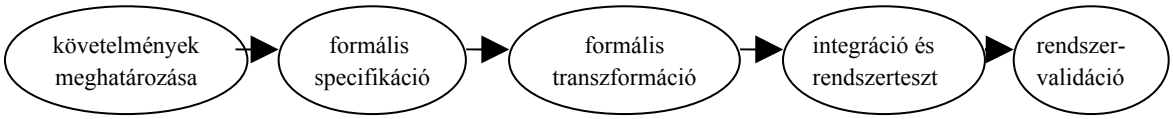
1. feltáró prototípus: a prototípusok segítségével a felhasználó és a fejlesztő közösen finomítja, pontosítja a követelményeket. A felhasználó kap egy működő verziót, ezek alapján pontosítja a követelményeket, fejlesztésnél ezeket figyelembe vesszük, így a következő verzió már közelebb jut a pontos követelményekhez. Így a verziósorozat egyes verzióiban egyre több felhasználói igény kerül beépítésre, míg a végső verzió már a tényleges követelményeknek megfelelő verzió lesz. Az első prototípus a rendszer azon részeit foglalja magába, melyek világosak, melyeknek a követelményei egyértelműek. A rendszer kevésbé ismert részei a későbbi prototípusokba kerülnek bele. Minden prototípus az előzőből következik, az előzőre épül.
2. eldobható prototípus: ennél a prototípusfajtánál a cél az, hogy a felhasználó segítségére legyünk a prototípussal abban, hogy a követelményeket pontosan meg tudja határozni. A felhasználó rendelkezésére bocsájtunk egy prototípust, így segítünk annak eldöntésében, hogy a rendszer mely részeit kell újradefiniálnunk, és ennek

megfelelően egy újabb prototípust hozunk létre, és az előzőt eldobjuk. Ebben a megközelítésben az első prototípusok a legkevésbé feltárt rendszerelemeket tartalmazzák annak érdekében, hogy a felhasználó pontosítsa a saját igényeit.

Az evolúciós modell előnyei közé tartozik, hogy a felhasználó igényeinek jobban megfelel, gyorsabban kap egy prototípust. Hamar ki tudja próbálni a verziókat, amelyek folyamatosan finomodnak, nem teljes alrendszert kell kipróbálni, csak bizonyos rendszerelemeket. A fejlődés bármely irányba úgy tud tovább menni, hogy a funkcionalitás egyre bővebb. Ez a modell viszonylag jól tervezhető és menedzselhető. Ezzel szemben a teljes szoftverfejlesztési folyamat nem látható, jellegénél fogva a szoftver struktúrája jelentősen romlik. Ezen modell központjában a gyors prototípusfejlesztés áll, amely maga után vonja a szükséges speciális ismereteket, speciális szakembereket, illetve speciális eszközöket.

4.3. Formális modell

A formális modell az 1970-es években alakult ki a vízesésmodell egy absztraktabb változataként. Strukturált megközelítésű modell.



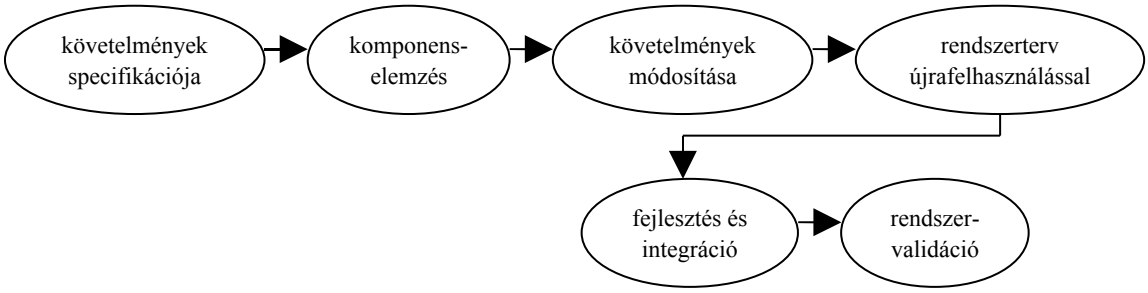
A követelményspecifikáció és a rendszer validáció ugyanaz, mint a vízesésmodell esetében. Ezen formális modell szerint azonban a rendszerspecifikációból matematikai eszközökkel formálisan generálunk működő programot. Ehhez az szükséges, hogy a rendszerspecifikációt is formálisan, absztrakt eszközökkel adjuk meg. Ekkor a lényeg a transzformációs folyamatban rejlik, amely a formálisan megadott követelményekből előállít egy adott nyelvű kódot.

Ennek a modellnek a segítségével megtakarítjuk a validálás folyamatát, mivel az egész modell a validáláson alapszik.

A modell előnye, hogy bizonyítottan helyes programot állít elő egy automatizált folyamat során. Hibája azonban, hogy a rendszerkövetelmények formalizálása kézzel kell, hogy történjen, azonban a rendszerkövetelmények matematikai eszközökkel való felírása általában nagyon nehézkes.

4.4. Újrafelhasználás (komponens) alapú modell

A modell lényege a nevében rejlik: újrafelhasználás. Ez azonban nem csupán kód-újrafelhasználást jelent, hanem tervezési szinten való újrafelhasználást is. A modell azon alapul, hogy léteznek rendelkezésre álló komponensek, melyeket újrafelhasználhatunk. Ez a következőképpen módosítja a folyamatot:



A komponenselemzés során megnézzük, hogy milyen kész termékek állnak rendelkezésre, ezeknek megfelelően módosítjuk a követelményeket, ezáltal a tervezés is a késztermékeken alapszik.

Ez a modell oda vezetett, hogy kialakult a komponens-orientált fejlesztés, amely azt jelenti, hogy eleve úgy történik a rendszer fejlesztése, hogy az újrafelhasználható komponens legyen.

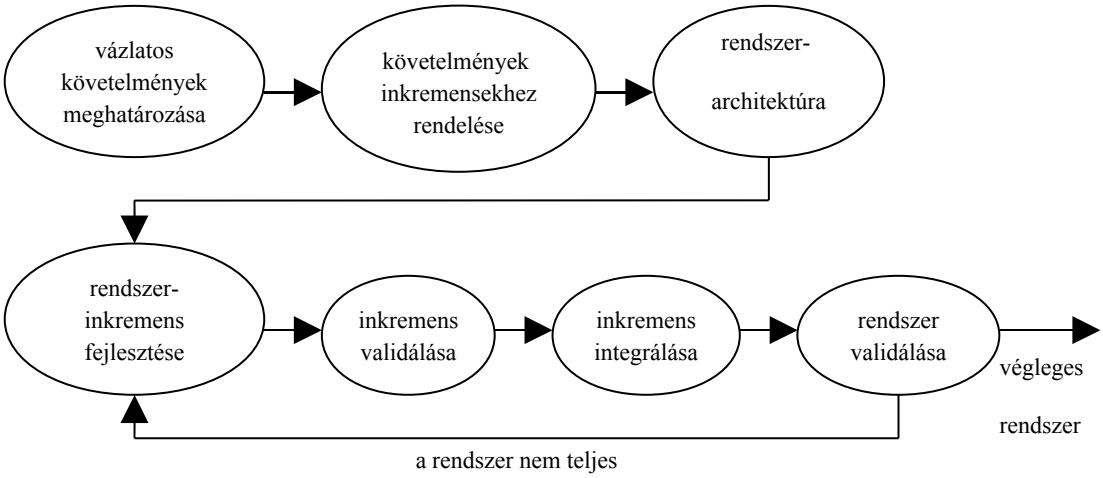
A modell előnyei közé tartozik, hogy kész komponenseket használunk fel, ezzel időt és pénzt takarítunk meg. A komponensek már léteznek, kipróbáltak, teszteltek, ezért feltételezhetjük, hogy nem hibásak vagy legalábbis minimálisan hibásak, így egyfajta biztonságérzetet nyújt, sok validációs lépést megspórolhatunk.

Hátrányai közé sorolható, hogy a már létező komponensekről megfelelő információval kell rendelkezni, tudnunk kell, hogy hol találhatjuk meg őket, valamint számolnunk kell azzal, hogy nem felelnek meg teljes egészében az igényeinknek. Problémát jelenthet még, hogy leromolhat a rendszer struktúrája.

4.5. Iteratív folyamatmodellek:

Arra épülnek fel, hogy a folyamat bizonyos fázisait ciklikusan megismétlik. Ez bármely fázisra vonatkozhat. Kétfajta modell létezik:

Az inkrementális modell az evolúciós modellből alakult ki oly módon, hogy azt az inkremensekkel való fejlesztéssel egészíti ki. A fejlesztés implementálás része kisméretű egységekben, inkremensekkel történik, a mindenkori rendszerbe inkremenseket építünk, inkremensekkel bővítjük, és ha szükséges, nyílván megismételjük az inkremenshez akár a specifikációs tervezés lépését is.



Az inkremensek hozzáillesztése ciklikus folyamat, melyet mindaddig folytatunk, míg úgy nem döntünk, hogy elkészült a rendszer. Az inkrementális fejlesztés lényege éppen ebben a ciklusban rejlik. Az iteráció általában az implementációs fázisra vonatkozik, de vonatkozhat a teljes folyamatra is.

Az inkremenseket önállóan fejlesztjük, tervezzük, implementáljuk, validáljuk, sőt adott esetben még a következményspecifikációt is megadhatjuk, megváltoztathatjuk inkremensenként. Az architektúrát változatlanul hagyva az inkremensek fejlesztése teljes életciklussal történik.

Inkremens fejlesztése történhet például vízesés modellel. Az inkremensek fejlesztése lehet párhuzamos, de az integrációnál csak egy inkremenst illeszthetünk be, erre vonatkozik a validáció is.

Az evolúciós modell előnyei jelentkeznek itt is. Mindig egy letesztelt, működő rendszert áll rendelkezésre. Ez az igazi nagy előnye. A fejlesztést azokkal az inkremensekkel kell kezdeni, amelyek a legfontosabb funkcionalitásokat valósítják meg.

Az inkrementális modell előnyei tehát, hogy az inkremensek kicsik, jól körülhatárolhatóak, adott esetben a követelmények egyértelműen specifikálhatóak, így ezekre önmagukban a legmegfelelőbb modell alkalmazható, például vízesésmodell. Az inkremensek fejlesztése történhet párhuzamosan. Kezdhetjük a fejlesztést a legfontosabb funkciókat realizáló inkremensekkel, és mivel az inkremensek kicsik, a felhasználó nagyon hamar kap egy nem eldobható prototípust, amelyet már tud használni. Tehát bizonyos funkciókat, a legalapvetőbb funkciókat tartalmazó rendszert nagyon hamar kap a felhasználó, a kevésbé lényeges funkciókat pedig majd a későbbiek folyamán beépítjük a rendszerbe. A rendszervalidálás mindig megtörténik, tehát a rendszer ilyen értelemben a beépített inkremensekkel önmagában egy használható részrendszer. A rendszerfejlesztés kockázata kisebb, mint az összes többi modellnél (a fejlesztések igen nagy része, több mint 50 százaléka sikertelen), mivel a rendszer validált, kis elemekkel, tehát van egy validált működő rendszer még akkor is, ha befullad a projekt, legfeljebb nem teljes funkcionalitással. Nagyon nagy a valószínűsége, hogy az első pár inkremensnél még nem fogy el a pénz, az idő, az ember, nem változik meg a környezet. Ezért a részsikeres befejezés valószínűbb ennél a fejlesztési modellnél. A fontosabb funkciókat implementáljuk először, ennek a következtében azokat a funkciókat a felhasználók rendszeresen tesztelik, a fontosabb funkciókat jóval többször használják, mint a kevésbé fontosakat. Így már a rendszerfejlesztés közben a hibák hamarabb

kiderülnek, a mindenkori részrendszerben valóban a legfontosabb funkciók a legteszteltebbek. Tehát egy robusztusabb rendszer fejlesztéséhez vezet ez a modell.

A modell problémáját a második és harmadik lépésben az inkremensek megtervezése jelenti: milyen funkciókat tegyünk egy-egy inkremensbe? Mit jelent egy inkremens? Ez a felosztás, az architektúra megtervezése az, ami nagyon nyűgös. A modell ezen alapszik, tehát ha az elején az architektúrát rosszul tervezzük meg, akkor gond van. Nagyon nehéz az inkremensek behatárolása, az elején eldöntjük, hogy melyek a fontosabb funkciók.

Az iteratív folyamatmodell másik fajtája a spirális modell. Ez a legkésőbb, 1988-ban kialakuló modell, az ún. Boehm-spirál. Teljesen más, mint az eddigiek. Középpontjába olyan dolog kerül, amivel az eddigi modellek nem foglalkoztak: a fejlesztés kockázati tényezőinek felmérése, a kockázatból származó problémákra való felkészülés, a sikertelen kimenetek, a rizikó csökkentése. Alapötlete, hogy nem ciklusokat csinálunk, hanem az életciklus minden fázisában csinálunk egy ciklust, és végigmegyünk minden fázison. Az első olyan modell, amely nagy hangsúlyt fektet a kockázatelemzésre.

Négy alaplépése van, amelyeket spirálisan ismételgetünk, tehát a következő spirálban ugyanaz a lépés az eddigieknél magasabb szinten ismétlődik meg.
1. lépés: A követelmények, pontosabban célok meghatározása. A kérdés: az adott spirális fázisban mit fogunk megvalósítani? Azonosítjuk a tevékenységeket, meghatározzuk a megszorításokat, a menedzselési tervet és azt, hogy milyen kockázati tényezők várhatók ebben a lépésben, valamint ezeket a kockázatokat hogyan fogjuk kezelni, milyen alternatívák lehetnek a kezelésre.
2. lépés: A kockázatelemzés, kockázatok felismerése, kockázatok tényezőinek megszűntetése, ezekre való felkészülés. Ekkor megtervezzük azokat a lépéseket, amelyek azt célozzák, hogy a kockázatokat el tudjuk kerülni. (Ha például a követelmények meghatározása kritikus pont, tehát úgy gondoljuk, hogy a teljes fejlesztésben nagyon nagy kockázatot jelent, rosszak a követelmények, akkor egy feltáró evolúciós modellt alkalmazunk ebben a lépésben, építenünk kell egy prototípus sorozatot a követelmények minél pontosabb meghatározása érdekében.) Fel kell tárni, hogy a fejlesztés adott fázisában milyen kockázatok vannak. Csökkenteni kell a kockázatokat minimálisra.

3. lépés: Az adott fázis végrehajtása és validálás. A második lépés után fejlesztési modellt választunk, lehetséges, hogy minden egyes spirálban más-más fejlesztési modellt alkalmazunk, adott esetben a kockázatanalízis eredményeként választható a fejlesztési modell. Ezek után hajtjuk végre a szokásos tervezés, implementáció, validálás lépéseket.
4. lépés: A teljes projekt, az eddigi fejlesztés és az eddigi spirálok áttekintése, elemzése, és a folytatásról való döntés olyan értelemben, hogy megállunk-e (a megvalósíthatósági esettanulmány után dönthetünk úgy, hogy befejezzük a fejlesztést). Amennyiben a folytatásról döntünk, a projektet tervezzük meg, pontosabban azt, hogy hogyan folytatódjon a projekt. Meg kell tervezni a fejlesztés következő spirálját.

A spirálmodell előnyei, hogy foglalkozik a kockázatokkal, sokkal szisztematikusabban foglalkozik a projekttel. Hátránya, hogy nem a legtriviálisabb modell.
Ez a modell nagy rendszereknél javallott, előnyei nagy rendszereknél jönnek ki.

4.6. Veterán Autó- és Motorkölcsönző

4.6.1. Fejlesztési modell

A példa alkalmazás fejlesztéséhez az evolúciós modellt választottam feltáró prototípussal. A felhasználó rendelkezésére bocsájtok egy működő verziót, ezek alapján lehetőséget teremtek arra, hogy pontosítsa a követelményeket. A felhasználó hamar ki tudja próbálni a verziókat, amelyek folyamatosan finomodnak. A választást segítette a modell sok előnye, mint például a jó menedzselhetőség, valamint a viszonylag jó tervezhetőség.

5. Követelmények

Mielőtt egy rendszer megvalósításához hozzákezdünk, pontosan tudnunk kell, hogy mit kell megvalósítanunk. Tisztában kell lennünk azzal, hogy mik a rendszerrel szemben támasztott elvárások, mik a rendszer követelményei. A követelmény egy rendszer szolgáltatásai és a rájuk vonatkozó megszorítások együttese.
A követelmények osztályozása több szempontból történhet.

Beszélünk felhasználói- és rendszerkövetelményekről.

A felhasználói követelmények azok az elvárások, melyet a rendszer későbbi alkalmazói támasztanak a rendszerrel szemben. Ezek a követelmények a funkcionalitásra vonatkoznak, azt specifikálják, hogy a rendszer mely szolgáltatásaira lesz a későbbiekben szükség, és ezek hogyan, milyen körülmények között, milyen elvárásokkal működjenek majd.

A rendszerkövetelmények ezzel szemben a rendszerre, mint egészre vonatkoznak, a teljes rendszerrel szembeni elvárásokat jelentik. Azok a követelmények, melyek általában, globálisan a rendszer egészére vonatkoznak, és függetlenek az egyes szolgáltatásokról.

Másik felosztás szerint beszélhetünk funkcionális-, nem-funkcionális- és szakterületi követelményekről.

A funkcionális követelmények: azok a követelmények, amelyek a szolgáltatásokhoz kapcsolódnak. A szemlélet egy kicsit azért más, mert a funkcionális követelmények felosztásánál azt mondjuk, hogy a rendszer bizonyos szituációkban bekövetkező reakcióit írják le, hogy hogyan reagál a rendszer bizonyos inputokra, bizonyos bekövetkező szituációkra. A funkcionális követelményeknél körül kell határolni a rendszert, tehát adott esetben a funkcionális követelmények közé felvesszük azt is, hogy a rendszertől mit nem követelünk meg, hogy mit nem kell tudnia a rendszernek.

A nem-funkcionális követelmények általában a teljes rendszerre kiterjedő követelmények, nem az egyes konkrét esetekre vonatkozó, hanem általános követelmények: időbeli megszorítások, korlátok, magára a fejlesztési folyamatra tesz korlátot. Adott esetben nem-funkcionális követelmény lehet az, hogy megadjuk, milyen fejlesztési modellt kell követni, vagy bizonyos szabványokat kell követni, a rendszernek konformnak kell lennie bizonyos szabványokkal. Ha egy rendszerkövetelmény nem teljesül, akkor a rendszer összeomolhat.

A szakterületi követelmények olyan követelmények, amelyek egy-egy adott szakterülethez kötődnek, tehát speciálisan az adott szakterület felhasználói igényeiből

következnek. Ezek lehetnek funkcionális követelmények, de nem az adott rendszerre vonatkozó, hanem általánosan a szakterületből következőek. Lehetnek általános, adott szakterületi rendszerekre vonatkozó megszorítások. Vonatkozhat például arra, hogy milyen hardveren, milyen eszközökkel kell kifejleszteni a rendszert, és, hogy milyen szabványok alapján. Nagyon kemény követelményeket szabhat meg. A nem-funkcionális követelmények nagyon általánosak, ezzel szemben a szakterületi követelmények nem azok.

A követelményeket természetesen megfelelően dokumentálnunk kell. Erre ma több módszer használatos: természetes nyelv, formanyomtatvány, diagram, matematikai megfogalmazás, hibrid megoldás.

A természetes nyelv illetve annak valamilyen strukturáltabb, formalizált változata jelen pillanatban elkerülhetetlen, a követelmények leírásának egy része minden esetben természetes nyelven történik. Előnye, hogy az adott nyelvet elviekben beszéli az a környezet, amely foglalkozik a követelményekkel. Hátránya azonban, hogy nem egyértelmű, terjengős, hosszú, nem szabványos, félreérthető, félremagyarázható ezen segít valamelyest a formalizált természetes nyelv.

A diagram a követelmények vizualizált megfogalmazása. Elég sokféle diagramot ismerünk, ma egyre nagyobb hangsúlyt kap. Előnyük, hogy áttekinthetőbbek, szabványosak, kevésbé félreérthetőek. Problémát jelent nagyméretű rendszerek követelményeinek áttekinthetősége: egy A4-es papíron lévő UML diagram még átlátható, öt A4-es lapon lévő már nem, néhány méteres diagram már képernyőn sem látható át, mert vagy a részleteket látjuk, de ekkor nem látjuk az egészet, vagy az egészet nem tudjuk megnézni egyben, mert nincs olyan eszköz, amely meg tudná jeleníteni.

A diagramok tehát szabványosak, egyértelműek, azonban a felhasználó gyakran nem érti őket, meg kell tanulnia. Éppen ezért általában a belső szabványok a formalizmust, a formanyomtatványt használják, amely egy strukturált szöveget jelenít meg, mely lehet természetes és nem természetes nyelvű.

Kevésbé elterjedt módon szokás még a követelmények megfogalmazásánál a matematikai megfogalmazást használni, mely a követelményeket valamilyen absztrakt eszközrendszer segítségével fogalmazza meg. Előnye, hogy egyértelmű, szabványos és rövid, viszont hátránya, hogy nem formalizálható, nagyon sok esetben érthetetlen a felhasználó számára.

Manapság szokás alkalmazni egyfajta hibrid megoldást, mely egy diagramorientált specifikáció, vizualizáció, rajt és egy természetes nyelvi fogalomszótár.

A rendszerkövetelmények megadásához nagyon gyakran rendszermodelleket használnak. Ezek a modellek grafikus reprezentációk; diagram segítségével adják meg a rendszerkövetelményeket. Ezek szolgálnak dokumentációként a tervezéshez, hangsúlyozván, hogy a rendszermodellek minden esetben absztrakciók. Ezek segítik formalizálni a követelményeket, képeznek egy hidat a felhasználó és a fejlesztő között. A rendszermodellek tehát a kifejlesztendő rendszert jelenítik meg valamilyen absztrakt szinten. Részben dokumentációs eszköz, részben pedig a felderítésben, követelmény-feltárásban használatosak. A rendszermodellek különböző oldalról szemlélik a rendszert: egyrészt közelíthetünk kívülről, vagyis megnézzük, hogy a rendszer milyen környezetben helyezkedik el, másik megközelítésben a rendszert, ha belülről szemléljük, akkor meghatározható a viselkedése, harmadrészt tekinthetünk a rendszerre strukturális szempontból belülről is, vagyis, hogy a felépítés az adatokat hogyan kezeli.

5.1. Környezeti modell

A követelmény-feltárás környékén alkalmazzák, és az architektúrális tervezésnél is alkalmazhatóak. A követelmény-feltárásnál meg kell húzni a vonalat, hogy mi tartozik a rendszerhez és mi nem. Ezt tudjuk lemodellezni ezzel a modellel. Meglehetősen egyszerű modell.

A környezeti modellek akkor alkalmazandóak, ha döntenünk kell a rendszer határairól. A funkcionális követelményeknél meg kell mondani, hogy mit nem tud a rendszer, rendszerkövetelményeknél le kell határolni azt. A rendszer határainak felderítése a környezeti

modellek elsődleges feladata. Ezek ún. architektúrális rendszermodellek. A követelménytervezés nagyon korai szakaszában kell alkalmazni, nagyon korán kell döntést hozni ezen modell alapján, hogy hol határoljuk le a rendszert. A környezeti modell nem mond semmit magáról a rendszerről belülről, de nem mond semmit a rendszer és a környezet között fennálló viszonyokról sem, azon felül, hogy kapcsolat van közöttük.

5.2. Viselkedési modell

A viselkedési modellből két irányzat alakult ki: az adatfolyam modellek, melyek tipikus strukturált modellek, valamint az állapotátmenet modellek, melyek tipikusan objektumorientált modellek.

Az adatfolyam modellek azt mutatják be, azt írják le, hogy az adatok, adatfolyamok hogyan mozognak, hogyan áramlanak a rendszeren belül, hogyan kerülnek feldolgozásra, mely elemtől mely elemhez kerülnek a feldolgozás során. Az adatfolyam modell előnye, hogy viszonylag könnyedén ráhúzhatóak a hétköznapi folyamatokra, tehát intuitívek, a felhasználó könnyen megérti.

Az állapotátmenet modell egy eseményvezérelt modell, mely az állapotátmeneteket írja le diagram segítségével.

A viselkedési modell problémái közé sorolható, hogy a rendszerben nagyon sokféle állapot, illetve nagyon sok adat transzformáció lehet, éppen ezért a diagram áttekinthetetlenné válik.

5.3. Adatmodellek

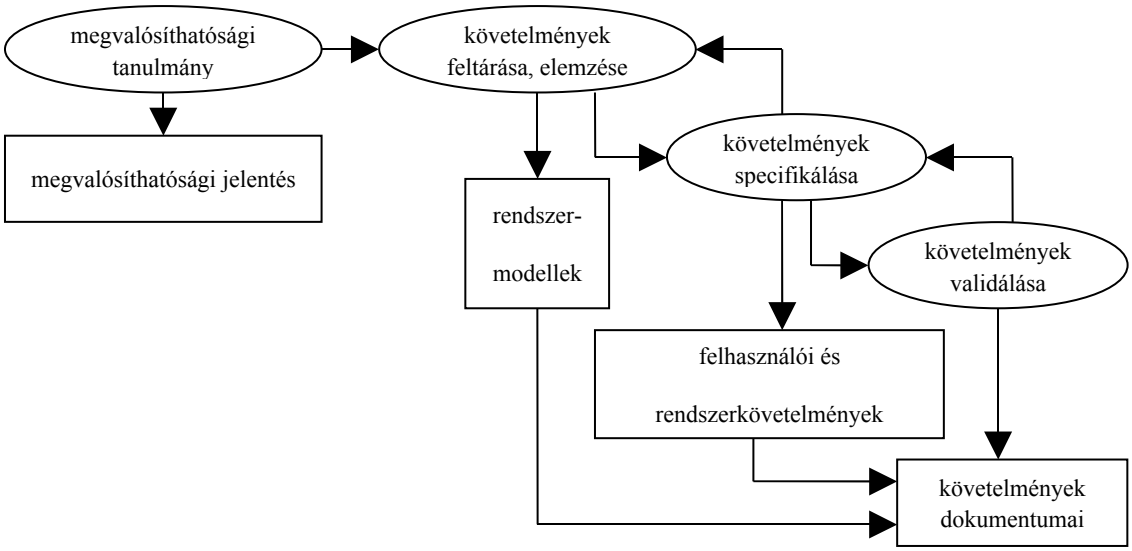
Az adatmodellek az adatintenzív rendszereknél lényegesek, ahol az adatok az adatbázisban tárolónak. Az adatmodellek kimondottan az adatok struktúráját írják le, az egyedek tulajdonságait, és az egyedek közötti kapcsolatokat modellezik. Az adatmodellek elsősorban a követelmény feltárás végén használatosak, a tervezéshez hidat képeznek, elsősorban a fejlesztőnek szólnak, nem pedig a felhasználónak.

5.4. Objektummodellek

Az objektummodellek objektumorientált paradigma mentén születtek meg. A mai technológiák mindegyike alkalmaz valamilyen objektummodellt. Az objektummodell leírásában tipikus modell az UML. Azért lett nagyon sikeres, mert a többi rendszermodellt le kell képezni a tervre, majd pedig azt implementálni, míg az objektummodellnél ez a leképezés elmarad, nincs szemantikai leképezés.

5.5. Követelménytervezés

A követelményeket is tervezni kell, kellő részletességgel szükséges feltárni őket, és megfelelő pontossággal rögzíteni különböző dokumentumokban, hogy az implementáláskor felhasználhatóak legyenek. A szoftverfejlesztés életciklusának ezt, az első nagy szakaszát szokás követelménytervezésnek, követelmény feltárásnak nevezni. Mindez négy nagy lépésből tevődik össze: a megvalósíthatósági tanulmány elkészítése, a követelmények feltárása és elemzése, a követelmények specifikálása és dokumentálása, a követelmények validálása.



A rendszerfejlesztés egy folyamat. A folyamat lépésekből áll, minden egyes lépés előállít egy eredményt, egy közbenső terméket. A folyamat lépéseit jelöljük ellipszissel, a folyamat eredményét téglalappal, a nyilak jelzik az átmenetet a lépések és a közbenső termékek között.

Először el kell készíteni egy megvalósíthatósági tanulmányt, amely felméri, hogy az elkészítendő rendszer ténylegesen hasznára válik-e majd üzleti szempontból a megrendelő számára. Ha a tanulmány pozitív eredményt ad, akkor fel kell tárni a követelményeket, elemezni kell a problémát, majd ezután ezeket a követelményeket valamilyen szabványos formára kell hozni, azaz specifikálni kell őket. Az utolsó lépésben a leírt követelményeket validálni kell, azaz meg kell állapítani, hogy a feltárt követelmények tényleg azt a rendszert írják-e le, amelyre a megrendelőnek szüksége van, tehát egy olyan rendszert definiál a specifikáció, amely ténylegesen a megrendelő kívánságai szerint fog működni.

5.6. Vízió

A szoftverfejlesztés akkor kezdődik, mikor valaki, általában egy cég, kitalál egy új elképzelést, egy víziót, melyet meg szeretne valósítani. Megfogalmazza a követelményvázlatot, és itt indul el a teljes követelménytervezési folyamat.

Egy szokásos esetet, mikor a megrendelő megfogalmazza az igényeit. A készülő alkalmazás vízióját csak ennyi alkotja, a megrendelő hozzávetőleges elképzelése, hogy milyen alkalmazást szeretne. Nagyon kevés olyan megrendelő van, aki pontosan elő tudja adni, hogy mit szeretne, mit vár el pontosan. Azonban pontosan specifikált követelmények nélkül nem lehet megfelelő alkalmazást írni. Ezért van szükség a követelményelemzésre, melynek során fel kell tárni a rendszer által nyújtandó funkciókat, a rendszer határait és a működés megszorításait.

5.7. Megvalósíthatósági tanulmány

A megvalósíthatósági tanulmány azt tartalmazza, hogy az adott vízió megvalósítható-e az adott cégen belül. A megvalósíthatósági jelentés egy olyan dokumentum, amely a rendszer

nagyvonalú leírását tartalmazza. Ezt szokás vázlatnak is nevezni. Ez a tanulmány arra szolgál, hogy ez alapján eldöntsük, hogy egyáltalán elindul-e a folyamat, lesz-e második lépés.

Az alapvető kérdések, amelyeket meg kell válaszolni ebben a tanulmányban: A kifejlesztendő rendszer támogatja-e az adott szervezet általános célkitűzéseit, munkáját, igényeit? Az adott rendszer megvalósítható-e a vízióban megfogalmazott időkeret, költségkeret és technológiai keretek között? A megvalósíthatósági tanulmányban alternatívákat kell leírni. A kifejlesztendő rendszer integrálható-e, beilleszthető-e a jelenleg használatban lévő rendszerbe?

A megvalósíthatósági tanulmány előállításához információt kell beszerezni. Ehhez egyrészt meg kell tudni, hogy kitől/kiktől szerezhető be a megfelelő információ, másrészt be kell tudni szerezni azt. Ez annyit jelent, hogy a vízió megbízójával kell konzultálni.

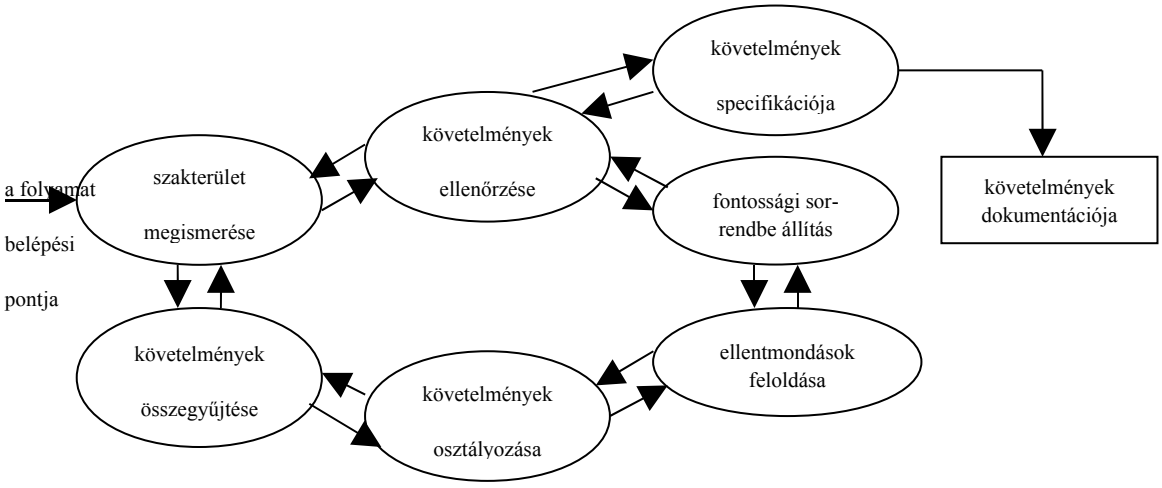
A megvalósíthatósági tanulmányban meg kell adni, hogy mit várok el a rendszertől és mit nem, majd dönteni kell, hogy elindítható–e a projekt. A döntés a felhasználó és a fejlesztő dolga.

5.8. Követelmények feltárása, elemzése

Ha a megvalósíthatósági tanulmány pozitív kicsengéssel ér véget, tehát arra a döntésre jutnak, hogy el kell kezdeni a rendszerfejlesztést, akkor a következő lépés a követelmények feltárása, elemzése. Ebben a szakaszban nagy szerepet játszanak az úgynevezett kulcsfigurák. Ide tartoznak mindazon személyek, akik valamilyen módon kapcsolatba kerülnek majd a rendszerrel (végfelhasználók, főnökök, menedzserek, rendszergazdák stb.). A követelmények feltárása és elemzése a kulcsfiguráktól beszerzett információk feltárását és elemzését jelenti.
A követelmények feltárása közben azonban különböző problémákkal kell szembesülnünk:

- A kulcsfigurák általában nem tudják, hogy mit várnak el a rendszertől, legalábbis nem tudják megfogalmazni. A mai felhasználók nagy része nem tudja, mit lehet elvárni egy számítógépes rendszertől, még akkor sem, ha már használják a számítógépet. Olyan elvárásaik vannak, amelyek vagy triviálisak, és már benne vannak a pillanatnyi rendszerben, vagy pedig megoldhatatlanok.

- A legnehezebb probléma még mindig a kommunikáció: a szakterület képviselői (a felhasználók vagy a megrendelő) és az információs rendszer fejlesztői között. Nyilván az adott szakterület implicit módon beépülő fogalmaival dolgoznak, amelyeket nem biztos, hogy explicit módon meg tudnak fogalmazni. Szót kell továbbá érteni az informatikusokkal, akik saját szakterületükön saját szakmai szlenggel dolgoznak, kiválóan elbeszélnek egymás mellett; már a mérnöki és a matematikusi informatikus világ sem ugyanazt a nyelvet beszéli az informatikában. A mérnökök más terminológiát, más fogalomrendszert használnak. Ez a dolog úgy oldódik meg, hogy mi informatikusok tanuljuk meg az adott szakterületet nyelvezetét, terminológiáját, specializálódunk, és így tudunk kommunikálni a megfelelő személyekkel.
- Különböző kulcsfiguráknak különböző követelményeik vannak, amelyek természetesen ellentmondanak egymásnak. A követelményelemzés legfontosabb feladata, hogy kiderítsék, hogy a követelmények valósak-e, a követelmények ellentmondásosak-e, ha igen feloldható-e ez az ellentmondásosság.
- A követelményeket nagyon gyakran külső körülmények is befolyásolják, pl. a politika. Politika alatt szokás érteni üzletpolitikát vagy vállalati politikát, de a nemzeti politikát is. A gazdasági környezet is megváltozhat, sőt ez állandóan, dinamikusan változik (pl. új kulcsfigurák léphetnek be), a követelmények feltárása és elemzése pedig adott esetben hosszú ideig tart. Megváltozhat a törvényi környezet is. Ezeket a változásokat mind-mind figyelembe kell venni.
-



Az ábrán a követelmények feltárásának folyamata látható. Ez egy iteratív folyamat, melyben minden lépés összefügg az összes többivel, iterálható. A megvalósíthatósági tanulmány után ennek eredményeképp megszületik a követelménydokumentáció.

A szakterület megismerése azt jelenti, hogy az informatikus, a fejlesztő kénytelen megismerkedni az adott szakterület fogalmaival, absztrakcióival, szokásaival. Lényeges, hogy a követelményeket a megrendelő és a fejlesztő együtt tárják fel olyan értelemben, hogy a megrendelő adja az összes információt, a többi a fejlesztő dolga. Nem várhatjuk el ugyanis, hogy a megrendelő az általa használni kívánt, kifejlesztendő rendszer specifikációját megadja.

A követelmények osztályozása, kategorizálása majd alapvetően befolyásolja a tervezést és a megvalósítást, illetve segít a következő két lépés megtételében (az ellentmondások feloldásában és a követelmények fontossági sorrendbe állításában).
Az ellentmondásokat fel kell oldani valamilyen módon. Ez nyilvánvalóan visszacsatolást és újraindulást jelent.

A fontossági sorrendbe állítás több szempontból is fontos. Miután már elviekben van egy ellentmondásmentes, kategorizált követelményrendszerünk, ott prioritásokat kell felállítani, mert ez egyrészt befolyásolhatja a további tervezést és implementálást, másrészt befolyásolhatja az időbeli ütemezést, a pénzeszközök elosztását.

5.9. Fogalomszótár

Mivel a különböző szakterületi szakértők az adott terület zsargonját használják, nehézkessé válik a folyamatok megértése a kívülálló követelményelemzők és tervezők számára. Ezért szükségessé válik a fogalomszótár összeállítása. Erre azért van szükség, mert így az alkalmazási szakterületen előforduló egyes szakkifejezések kifejtésre kerülnek. Tehát a megrendelő elmagyarázza, hogy a szakterület egyes fogalmai pontosan mit is jelentenek, milyen a rendszer szempontjából fontos adatokkal rendelkeznek, illetve mely üzleti folyamatokkal vannak kapcsolatban.

5.10. Követelmények összegyűjtése

A követelmények összegyűjtésére több technika született. Létezik úgynevezett nézőpont-orientált megközelítés, megpróbálják a kulcsfigurákat különböző típusokba sorolni, akik más-más oldalról szemlélik a kifejlesztendő rendszert, más-más követelményeik, prioritásaik vannak, más nézőpontokat képviselnek. Ilyen értelemben beszélhetünk a kulcsfigurák azon csoportjától, akik belülről látják a rendszert, másik csoport, akik kívülről, feketedobozként tekintenek a rendszerre.

A követelmények összegyűjtésének másik módja a forgatókönyvek alkalmazása. Ez a módszer nagyban hasonlít a filmek forgatókönyvéhez, leírjuk bennük, hogy egy adott probléma megoldása milyen konkrét lépéseken keresztül végezhető el. Ezen leírások alapján a kulcsfigurák képesek megérteni, hogy az adott szituáció megvalósítása közben hogyan érintkeznek majd a rendszerrel, és emiatt a felmerülő problémáikat és kritikáikat is meg tudják fogalmazni. Az így felmerülő információkat a követelménytervezők nagyon jól tudják hasznosítani a rendszerkövetelmények finomításakor. Tehát a forgatókönyvek segítenek a már eddig körvonalazott és dokumentált követelményeket. A forgatókönyv nem más, mint egy leírt interakció-sorozat, mely rögzíti, hogy a forgatókönyvben szereplő kulcsfigurák milyen lépéseket tesznek, és ezeknek milyen hatásuk van a rendszerre. Minden forgatókönyv egy vagy több lehetséges interakciót ír le. Az elkészítése ezen interakciók ezen interakciók körvonalazásával kezdődik, a feltárás alatt további részletekkel bővítjük, hogy végül az interakció teljes megírása megszülessen. A részletezés egy egyszerűbb módja, ha minden egyes használati esethez megadunk egy vagy több forgatókönyvet, amelyben felsoroljuk, hogy a funkció milyen, az adott kulcsfigura és az alkalmazás között lezajló párbeszédet igényel. A forgatókönyvek megadásakor célszerű a párbeszédre, az interakciókra helyezni a hangsúlyt. A forgatókönyvek így olyan pontokból állnak, amelyek vagy a rendszer felé történő adatszolgáltatást vagy a rendszer adatszolgáltatását, például eredményközlést írja le. Először a tipikusnak tekintett forgatókönyveket vegyük fel, majd ezután térjünk ki a speciális, például a hibás esetek párbeszédére. Egy forgatókönyvnek a következőket kell tartalmaznia: a rendszer állapotát az induláskor a forgatókönyv elején, az interakció normál menetét, a kivételes eseteket, valamint ezek kezelését, vagyis a kivételkezelést, az interakcióval egyidőben történő tevékenységeket, ha ezek értelmezhetőek, valamint a rendszer állapotát a

folyama végén. Két típusát különböztetjük meg: az esemény forgatókönyvet, mely az interakciót arra hegyezi ki, hogy milyen belső eseményekre hogyan reagál a rendszer. A forgatókönyvek másik típusa a használati eset, mely esetén a külső szereplők és a rendszer közötti eseményekre vannak az interakciók kihegyezve. A forgatókönyvek nagy előnye, hogy informális leírások, így könnyen érthetőek a kulcsfigurák számára, illetve elkészítésük során a követelményelemzők együtt dolgozhatnak a kulcsfigurákkal. Így könnyen azonosíthatóak a szükséges forgatókönyvek, és azok részletei Ezek a dokumentumok elsősorban szövegese, de szükség esetén kiegészíthetőek diagramokkal, képernyőelemekkel és magyarázó folyamatábrákkal. A követelmény összegyűjtésére nyújt lehetőséget az etnográfia, amely egy megfigyelésen alapuló technika. Ez a technika vissza fog térni a rendszerfejlesztés más fázisaiban is. követelményfeltáró elmegy a helyszínre és megfigyeli, hogy mi, hogyan zajlik. Ez alapján határozza meg a követelményeket. Időigényes, de sok előnye van. A tényleges folyamatot tudják megfigyelni. Meg lehet figyelni, hogy a folyamatok hogyan hatnak egymásra és, hogy a követelmények hogyan épülnek egymásra. Előnye, hogy ennek segítségével olyan követelmények is feltárhatók, amelyeket a kulcsfigurák nem tudnak elmondani, explicit módon megfogalmazni, mert adott esetben nem is tudnak róla, vagy nem is tudatosítják. Hátránya, hogy nagyon időigényes, több órát is ott kell ülni és figyelni, hogy hogyan történnek a folyamatok, események, amelyek követelményeit majd össze kell gyűjteni. Másik hátránya, hogy nagyon nagy tapasztalatot igényel.

Végül az utolsó lehetőség a követelmények összegyűjtésére az interjúkészítés, amely során a kulcsfiguráknak valamilyen módon célzottan teszünk fel kérdéseket, és így próbáljuk meg megfogalmazni a követelményeket.

A követelményeket ezután valamilyen szempont szerint osztályoznunk kell. A kategorizálás már a követelményelemzés része, mely során ki lehet szűrni a másképpen megfogalmazott, de azonos követelményeket. A kategorizálás után a követelményosztályokat priorizálnunk kell. Az elemzés során ciklikusan finomítjuk a követelményrendszert megfelelő visszacsatolással. Nagyon fontos ez a ciklikusság, mivel ezzel a későbbiekben sok problémát előzhetünk meg.

5.11. Követelmények ellenőrzése, validálása

A követelmények validálása annyit jelent, hogy az elkészített dokumentumokat megvizsgáljuk, hogy tényleg azokat a funkcionalitásokat fedi le, amelyeket a megrendelő kíván majd az elkészített szoftvertől. Ez ténylegesen ellenőrzési fázis, a teljes követelmény együttest visszacsatoljuk a megrendelőhöz. A validálási folyamat teljes egészében átfedi az elemzést, mert elsődleges célja az, hogy megtalálja a követelményekkel kapcsolatos problémákat, így folyamatos ellenőrzés szükséges. Bár sokan elhanyagolhatónak tartják ezt a részfolyamatot, azonban egy valódi szoftverfejlesztési feladatban szükséges ez a lépés, mivel ha eleve hibás követelmények alapján implementálnánk egy részfunkciót a rendszerben, majd ezt csak akkor vennénk észre, mikor már az adott programrész működik. Ekkor egészen a tervekig vissza kellene nyúlni, és megváltoztatni azokat.

A következményvalidálást a felhasználók és a fejlesztők együtt végzik. Ez tehát egy ellenőrzési lépéssorozat, amely a követelménydokumentáción hajtandó végre.

A validitásellenőrzés annak ellenőrzésére szolgál, hogy a rendszer minden funkciót, szolgáltatást produkál-e, a majdan kifejlesztendő rendszer követelményei között fel van-e sorolva minden funkció, amelyre szüksége lesz a rendszer felhasználóinak. Ez a lépés azért lényeges, mert vannak ellentmondások, ezeket fel kellett oldani, mire ehhez az iterációs lépéshez érünk, elviekben az ellentmondások ki vannak szűrve, azonban az ellentmondások megszűntetésénél kompromisszumokat kellett kötnünk. Azt kell ellenőriznünk, hogy ezek után meg van-e még a rendszer teljes funkcionalitása, valóban nyújtja-e azokat a szolgáltatásokat, amelyek szükségesek a rendszer felhasználói számára. A validitásellenőrzés lépései tehát:

- ellentmondás mentesség a teljes dokumentációra vonatkozóan
- teljességellenőrzés: itt a problémát az jelentheti, hogy nem tudjuk, hogy mihez képest teljes vagy nem teljes a rendszer
- megvalósíthatóság: eljutottunk oda, hogy vége van egy folyamatnak, létrejött a követelménydokumentáció, mint közbenső termék. Ugyan a megvalósíthatósági tanulmány egy vázlatos elképzelés alapján adott valamilyen megvalósíthatósági elképzelést, amire igent mondtunk, most már megvannak a pontos követelmények,

ellenőrizni kell a dokumentáció ismeretében, hogy a rendelkezésre álló infrastruktúra, időkeret, pénzkeret között megvalósítható-e a rendszer kifejlesztése, a további lépések.

- verifikálhatóság: a követelményrendszer alapján fog elkészülni a rendszer. Amikor a fejlesztő a kész információs rendszert átadja, akkor neki azt kell bizonyítani, hogy a rendszer, megfelel a követelménydokumentációban leírtaknak. Itt a verifikálás azt jelenti, hogy azt kell megnézni a követelménydokumentációban, hogy a majdan kifejlesztendő szoftver esetén hogy lehet majd bizonyítani azt, hogy teljesíti a követelményrendszert.

A validálásra különböző technikák léteznek:

A felülvizsgálat során a már specifikált és rögzített követelményeket egy emberekből álló csoport ellenőrzi. Ez egy manuális tevékenység, a megrendelői és a felhasználói oldalról az emberek végignézik a követelménydokumentációt. Ez lehet informális és formális felülvizsgálat. Az informális felülvizsgálat a megrendelő oldalán lényegesebb, a megrendelő végfelhasználói, kulcsfigurái közül minél több személlyel meg kell ismertetni a követelmény-dokumentumot és ki kell kérni a véleményüket. Szembesítjük a megrendelőt az általa elmondott követelményekkel, ez egyfajta visszacsatolás, hogy komolyan gondolták-e. A formális technika esetén megalakul egy felülvizsgálati csoport, amely újból végrehajtja az iteráció lépéseit, nem gyűjti, hanem elemzi a köve követelményeket. A formális felülvizsgálat esetén középpontba kell állítani a verifikálhatóságot.

Validáláshoz generálhatunk teszteseteket. A tesztesetek az implementációhoz, a rendszerfejlesztők fejében pedig követelményekhez kötődnek. Ezek a tesztesetek a tesztelésnél ismét előjönnek, itt most az a lényeges, hogy hogyan tudunk teszteseteket generálni a követelménydokumentum alapján. Ha ez nehezen megy, akkor probléma van a követelményekkel, mert ha a teszteseteket nehéz generálni, nehéz is lesz implementálni.

Speciális esetben lehetséges az automatikus validáció. Ha a követelményspecifikáció leírásához bizonyos rendszer-modelleket, ill. a specifikációhoz bizonyos formális leíró eszközrendszert alkalmazunk, akkor automatikus követelményellenőrzés is lehetséges.

5.12. Követelmények menedzselése

Azzal, hogy létrehoztunk és validáltunk egy követelménydokumentumot, egy pillatnyi állapotot rögzítettünk, azonban a világ dinamikusan változik, így aztán a követelmények is időben változnak, a rendszereknek van egy evolúciója. Egy nagyméretű rendszer fejlesztése hosszú, alapból garantálja, hogy mire elkészül a rendszer, addigra megváltoznak a követelmények is. A követelmények menedzselésénél erre kell felkészülni. Ilyen környezeti változások például az információtechnológiai környezet változásai: új platform, új hardver, új technológia.

A követelmények menedzselésekor arra kell felkészülni, hogy hogyan tudjuk konzisztens módon a követelmények változásait dokumentálni, megtervezni, hogy a változást követni tudjuk. Erre a technológiában (pl. .NET platformon, OO platformokon) viszonylag jól automatizált dokumentációs eszközök állnak rendelkezésünkre. Ezek biztosítják a követelmények azonosítását (tehát a követelményeket valamiféle azonosítóval kell ellátni, pl. sorszámozással), változtatáskezelést (a megváltozott követelményt át tudjuk vezetni az új dokumentációba), nyomonkövethetőséget.

Változtatáskezelés esetén egyértelmű kell, hogy legyen az, hogy egy követelménye megváltoztatása milyen hatásokkal jár. Változás esetén meg kell vizsgálni a kihatásokat. Egy követelmény megváltoztatása egy olyan lépéssorozat, melyet bármelyik fázisban megtehetünk. Döntést kell hozni, hogy a változtatást megengedhetjük–e, vagy anélkül, hogy folytatjuk tovább a projektet. Ha igen, akkor a változást át kell vezetni az összes fázison.

Ezzel tulajdonképpen elérkeztünk a szoftverfejlesztés első nagy fejezetének a végére. Most nézzük meg, hogy hogyan néz ki ez az első fejezet egy konkrét példa, egy Veterán Autó- és Motorkölcsönző rendszer esetén.

5.13. Veterán Autó- és Motorkölcsönző

5.13.1. Vízió

Vegyük a Veterán Autó- és Motorkölcsönző céget. Ez a cég már jó ideje sikeresen működik a piacon, szolgáltatásaik, vagyis a veterán járművek kölcsönzése széles körben elterjedt. A veterán csodák szerelmesei kölcsönözhetnek egy-egy járművet egy- vagy akár több napra is. A cég nyílván tartja a birtokában lévő veterán autó és motor fontosabb adatait, az ügyfelei adatait, valamint az egyes kölcsönzések adatait. Természetesen tudniuk kell, hogy melyik járműre milyen intervallumban van már kölcsönzési igény leadva, tudniuk kell, hogy ki adta le az adott igényt, és így természetesen megfelelő információval kell rendelkezniük arról is, hogy mely autók illetve motorok állnak az ügyfelek rendelkezésére az egyes időpontokban.

Összefoglalva tehát van egy sikeresen működő cég, a maga jól bevált üzleti folyamataival. Azonban a piacon egyre nagyobb számmal jelennek meg a versenytársak, akik kedvezőbbnél kedvezőbb ajánlatokkal csábítják az ügyfeleket. Éppen ezért a szóban forgó cégnek is ki kell találnia valamit, hogy ügyfelek tömegét magához vonzza biztosítva ezzel a további fennmaradás feltételét. Napjainkban szinte minden a számítógép és az Internet körül forog. Az élet minden területén használják. Már nem szükséges elmennünk a boltba, hogy egy adott termékhez hozzájussunk, elég bekapcsolni asztali számítógépünket, esetleg laptopunkat, fellépni az Internetre, és megvásárolni a kívánt terméket az online áruházak valamelyikéből. Ráadásul nem szükséges, hogy az adott termék az adott boltban raktáron legyen, mi leadjuk a rendelési igényünket, az üzlet beszerzi nekünk a megfelelő árut, és már postázzák is a címünkre. Az Internet világa tehát óriási lehetőségeket tartogat a különböző áruházak, cégek, szolgáltatók számára.

Éppen ezért a Veterán Autó- és Motorkölcsönző cég vezetőjében megfogalmazódik a gondolat, hogy az Internet felhasználói számára is lehetőséget biztosítsanak különböző járművek kölcsönzésére, ráadásul otthonról, kényelmesen. Ez azért fontos, és gyümölcsöző gondolat, mert az adott ügyfélnek be sem kell fáradnia a cég telephelyére, otthon nézegetheti

a veterán csodák fényképeit, beleképzelheti magát az egyes járművekbe, és pár kattintással elérheti, hogy a kiválasztott darab a megfelelő időpontban a rendelkezésére álljon.

5.13.2. Követelmények

A kiválasztott alkalmazás egy Veterán Autó- és Motorkölcsönző rendszer lesz. A szoftverfejlesztés kezdetén el kell döntenünk, hogy mit is szeretnénk pontosan megvalósítani a fejlesztés során, melyek azok az igények, elvárások, követelmények, melyeknek a fejlesztendő rendszernek meg kell felelnie.

Követelmények az online járműkölcsönző rendszerrel szemben:

- Regisztráció: lehetőség legyen a felhasználók számára regisztrálni a rendszerbe, és ezzel bizonyos kitüntetett jogokat szerezni. Mindenképpen lennie kell egy adminisztrátornak. Ehhez szorosan hozzátartozik, hogy a felhasználók tudják a saját jelszavukat változtatni.
- Adatbázis karbantartása: tudnunk kell az adatbázisba új járműveket felvenni, ha új autó vagy motor érkezik a céghez, illetve tudnunk kell törölni is járműveket, ha valamilyen oknál fogva azt már nem lehet kölcsönözni többé.
- Böngészés: biztosítanunk kell a veterán járművek közötti böngészést minden felhasználó számára, még a nem regisztrált ügyfelek részére is, hiszen később ezekből a látogatókból is ügyfelek válhatnak.
- Kölcsönzés: a regisztrált felhasználók részére biztosítani kell a kiválasztott járműre vonatkozó kölcsönzés igényének adatbázisban való rögzítésének lehetőségét. Természetesen, ha egy adott jármű a kívánt időszakban már le van foglalva, elvárjuk a rendszertől, hogy felhasználóbarát módon értesítse a felhasználót erről. Idesorolandó, hogy az adminisztrátornak természetesen lehetősége nyílik a leadott megrendelések megtekintésére.
- Használhatóság: fontos követelmény, hogy az alkalmazás ne legyen túlbonyolítva, legyen egyszerű, s funkcionális. Az egyszerű felhasználó számára is legyen átlátható, egyértelmű és könnyen kezelhető.
- Hordozhatóság: az alkalmazásunk platform- és rendszer független legyen.
- Validitás: a programunk lehetőleg feleljen meg a HTML és egyéb szabványoknak. Különböző böngészőkben is ugyanúgy jelenjen meg a weblapunk.

- Adatvédelmi és biztonsági követelmények: programunk feleljen meg a biztonsági és adatvédelmi előírásoknak.

5.13.3. Fogalomszótár

- nem regisztrált ügyfél: olyan személy, aki nem rendelkezik regisztrációval, és a cég honlapját meglátogatva böngészhet a kölcsönözhető veterán autók és motorok között, megtekintheti ezek adatait, valamint lehetősége nyílik a regisztrációra.
- regisztrált ügyfél: olyan személy, aki rendelkezik regisztrációval. A regisztrált ügyfelek a rendszerbe való belépést követően a veterán járművek böngészése mellett rendelkeznek a kölcsönzés jogával is, vagyis a kiválasztott járművet a kívánt időben lefoglalhatják. Lehetőségük nyílik emellett a jelszavuk módosítására.
- adminisztrátor: olyan személy, aki magasabb fokú jogosultsággal rendelkezik az egyszerű regisztrált felhasználónál. Megtekintheti az egyes járművekre leadott kölcsönzési igényeket, törölhet közülük. Lehetősége nyílik az egyes felhasználók karbantartására, így akár maga mellé létrehozhat másik adminisztrátort is. Ő az a felhasználó, aki új járművet vehet fel az adatbázisba.
- regisztrációs feltétel: olyan szerződés, amely kimondja, hogy az ügyfél az általa választott jármű használati jogát az adott időintervallumra megkapja, ha ezért a megfelelő díjat kifizeti. Amennyiben a jármű a kölcsönzés közben meghibásodik, az ügyfél csak a meghibásodás napjáig köteles kifizetni a kölcsönzési díjat. Azonban ha a jármű a kölcsönzés közben megsérül, az ügyfél köteles megjavíttatni a járművet a saját költségén.

5.13.4. Forgatókönyv

<u>Regisztráció:</u>

- kiinduló feltétel: a felhasználó a böngészőjében betöltötte a honlapot
- normális működés: a felhasználó a regisztrációs linkre kattintva a regisztrációs oldalra kerül. Ott ki kell töltenie a megfelelő adatokat, majd pedig a regisztráció gombra kattintva a rendszer ellenőrzi, hogy minden mező ki van-e töltve, és ha igen, akkor az

adatbázisba felveszi az adott felhasználót, és átirányítja a felhasználót a bejelentkezés lapra.

- hibás esetek: ha a regisztrációs űrlap nem megfelelően lett kitöltve, akkor a rendszer üzenetet küld a felhasználónak a hiba jellegéről. Ha az adatbázisban már szereplő felhasználói nevet ad meg a felhasználó, akkor a rendszer értesíti, hogy ilyen felhasználói névvel már regisztráltak korábban.
- rendszerállapot: sikeres regisztráció esetén a rendszer elmenti az adatbázisba az adott felhasználó adatait.

Kölcsönzés:

- kiinduló feltétel: a regisztrált felhasználó be van jelentkezve a rendszerbe.
- normális működés: a felhasználó böngészés közben a kiválasztott jármű neve mellett szereplő „Kibérelem” linkre kattint, így eljut a Kölcsönzés oldalra, ahol a kiválasztott autót vagy motort a kívánt időpontra lefoglalhatja. Ezen az oldalon meg kell adni, hogy mettől meddig szeretné kölcsönvenni a járművet év, hónap, nap pontossággal. Ezután a megrendel gombra kattintva a rendszer ellenőrzi, hogy helyes intervallumot adott-e meg az ügyfél, és helyes válasz esetén rögzíti a rendelést az adatbázisban az adott felhasználó azonosítója mellett.

– hibás esetek: ha a felhasználó hibás intervallumot adott meg, akkor a rendszer elküldi a megfelelő hibaüzenetet a felhasználó számára. Ilyen hibás intervallum lehet például, hogy 2010.01.01-től szeretnénk foglalni 2009.12.30-ig. Ekkor a rendszer azt a hibaüzenetet küldi, hogy a kezdőév nem lehet később, mint a záróév.
– rendszerállapot: sikeres kölcsönzés esetén a rendszer elmenti az adatbázisba az adott rendelést az adott felhasználó azonosítójával.

6. Tervezés

A szoftverfejlesztés első nagy fejezete, a követelményelemzés már lezárult, a rendszer megvalósításához szükségesnek vélt követelményeket feltártuk, és rögzítettük a megfelelő dokumentumokban. Ha ezek a lépések megfelelően történtek, akkor ennek alapján már hozzákezdhetünk a rendszer megtervezéséhez. A szoftvertervezés lényege a szoftver logikai szerkezetére vonatkozó döntések meghozatala, melyet valamilyen módon le kell írnunk, hogy a fejlesztők számára megvalósítható legyen. Ez a leírás történhet valamilyen logikai modell segítségével, mint például az UML vagy pedig egy informális jelölésrendszer segítségével is reprezentálhatjuk a rendszer terveit.

6.1. Modulokra bontás

Az egész rendszerünk túlságosan nagyméretű ahhoz, hogy teljes egészként kezeljük, éppen ezért szükségünk van a rendszer modulokra bontására. Ennek célja a tervezési folyamat egyszerűsítése, áttekinthetőbbé tétele, a fejlesztési feladatok részekre való szétosztása. A modulok nem következnek a követelményekből, a modulok szerepe ténylegesen a tervezésben jön el. A tervezés eredményeként a szoftver modulok specifikációja, valamint a köztük lévő interfészek, illetve kapcsolódási folyamatok terve készül el.

A modulokra való bontás után következik a modulok elkészítése és tesztelése. Ebben a szakaszban történik meg az egyes modulok forrásnyelvi kódjának megírása, ezt követően pedig az elkészült modulok önálló tesztelése. A tesztelési folyamatok előzetesen megtervezendők. A tesztelés már integráns része a szoftver verifikációs folyamatának, amelyben azt döntjük el, hogy egy-egy modul megfelel-e a specifikációjának. Ehhez az inputot a modulok terve szolgáltatja.

Miután mindegyik modul átment a tesztelésen, a teljes rendszer összeintegrálására kerül sor. A gyakorlatban kétféle megközelítés terjedt el erre vonatkozóan. Az első az integrálás egyenkénti bővítéssel és teszteléssel, az inkrementális tesztelés. Ekkor egy kiindulási, minimális számú modulhoz egymás után illesztjük a bővítésként szolgáló modulokat. Minden egyes bővítés után külön teszteljük az addig összeállt komplexumot. A megoldás előnye az, hogy az újabb hibák egy-egy bővítés során léphetnek be nagy valószínűséggel, s így könnyebb felfedni őket a bonyolultság lépcsőzetes növekedésével. A másik megközelítésben az összes modult egyszerre rakjuk össze, és a teljes komplexumra hajtjuk végre a tesztelést, ez az együttes tesztelés. Ez a megoldás azon a feltevésen alapul, hogy az előzetesen kitesztelt modulok már nagy valószínűséggel együtt is helyesen fognak működni.

6.2. Tervezés újrafelhasználással

Az egész ipari, mérnöki munka egy újrafelhasználás-orientált tervezési munka. Amikor egy cégnél új autót terveznek, az nem azt jelenti, hogy az egész autót újratervezik, hanem bizonyos terveket eleve újrafelhasználnak.

Az újrafelhasználás már a számítástechnika kezdetén felmerül, valahol a „szoftverelemek újrafelhasználása" címszó alatt, majd valamikor a 90-es évek közepén a tervezési folyamatokban is megjelenik. Azóta beszélünk erről, hogy tervezés újrafelhasználással. A szoftverelemek újrafelhasználásától eljutunk a tervelemek újrafelhasználásáig.

Kétféle közelítés létezik. Az egyik az alkalmazkodó újrafelhasználás: ez szoftver, tehát ténylegesen a programozás környékén jelenik meg. Ez a programozásnál annyit jelent, hogy felfedezzük, hogy léteznek a kívánt követelményeknek eleget tevő szoftverelemek, és ezeket fel tudjuk használni. Az alkalmazkodás annyit jelent, hogy implementálás közben fedezzük fel az elemeket, és implementáláskor illesztjük be a programba. Hasonló a helyzet tervezésnél: felfedezzük, hogy léteznek a tervünkbe illeszthető tervelemek. A másik megközelítés a módszeres újrafelhasználás: ez már a tervezésnél jelenik meg az újrafelhasználás tervezésének a specialitásaként, amikor is a tervezési folyamatot úgy alakítjuk, hogy keresünk újrafelhasználható tervelemeket, tehát a tervezést eleve úgy képzeljük el, hogy újrafelhasználható tervelemek köré építjük fel. Ennek a legalsó szintje az, hogy a tervezés folyamán újrafelhasználható szoftverelemeket, komponenseket keresünk és beépítjük azt a tervbe.

Előnyei:

- megbízhatóság: az újrafelhasznált elemek, a komponensek adott esetben már régóta működnek, ezeket már néhányszor vagy sokszor újrafelhasználták, különböző platformokon és környezetekben kerültek kipróbálásra, tesztelésre, a bennük levő hibák már kiderültek, kijavították őket, tehát egy megbízhatóan működő szoftverelemet tudunk beépíteni.

- kockázat: egy komponens felhasználásakor csökken a kockázat, ez igaz, minél nagyobb a komponens. Egy megbízható szoftverelemet építünk be, kisebb annak az esélye, hogy a rendszerünk rosszul fog működni. Annak kockázata pedig, hogy nem fog elkészülni az adott komponens, nagyjából nulla, mivel már ott van.
- szakemberek hatékonyabban alkalmazhatók ugyanis ekkor az adott komponensek vonatkozásában specialistákról beszélhetünk, hosszú ideig ugyanazzal a komponenssel foglalkozik, szakértelme beleépül a komponensbe, ezt a szakértelmet tudjuk felhasználni a fejlesztésünkben, nem pedig egy újra induló, adott esetben a területen ismeretlen, egyébként jó informatikai szakembert tudunk alkalmazni.
- a komponensek szabványosíthatók: például a felhasználói felületek szabványarchitektúrával rendelkeznek, mögöttük szabványos újrafelhasználható komponensek vannak.
- a rendszer kifejlesztési ideje rövidebb, rövidül a fejlesztés, a tesztelés, sőt a tervezés is.

Problémák:

–a komponensekkel kapcsolatban van három dolog, amely nélkül nem működne az egész:

- meg kell tudni találni a felhasználható komponenseket
- bizalom kérdése: aki a komponenst használni akarja, el kell hinnie a komponensekkel kapcsolatos információkat, bíznia kell benne, hogy igazak. A komponensek tipikusan olyanok, hogy forráskódjuk nem publikus, nem áll rendelkezésünkre.
- a komponenseknek dokumentáltaknak kell lenniük

- általában nő a karbantartási költség, mert a rendszerek változnak, a rendszereket módosítani, változtatni, karbantartani kell
- a komponensek karbantartása: a komponenseknek is kellene, hogy legyen evolúciója, ami nem biztos, hogy működik, mivel lehetséges, hogy a komponenst megalkotó cég már nem is létezik.

6.3. Veterán Autó- és Motorkölcsönző

6.3.1. Tervezés - Modulokra bontás

Mivel az egész rendszerünk túlságosan nagyméretű ahhoz, hogy teljes egészként kezeljük, ezért szükségünk van a rendszer modulokra bontásához.

A Veterán Autó- és Motorkölcsönző rendszer moduljai a következők:

- adatok módosítása: a felhasználó a saját adatait módosíthatja
- böngészés: a cég különböző veterán járműveinek áttekintése. A járművek adatainak, illetve fényképeinek megtekintése.
- bérlés: a kiválasztott autó illetve motor bérlése
- rendelések szerkesztése: az adminisztrátor az összes foglalási igényt megtekintheti, illetve szükség esetén törölheti azokat.
- felhasználók karbantartása: az adminisztrátornak lehetősége van a felhasználók a jogainak szerkesztéséhez
- új jármű felvitele: az adminisztrátor a cég újonnan beszerzett járműveit rögzítheti az adatbázisban

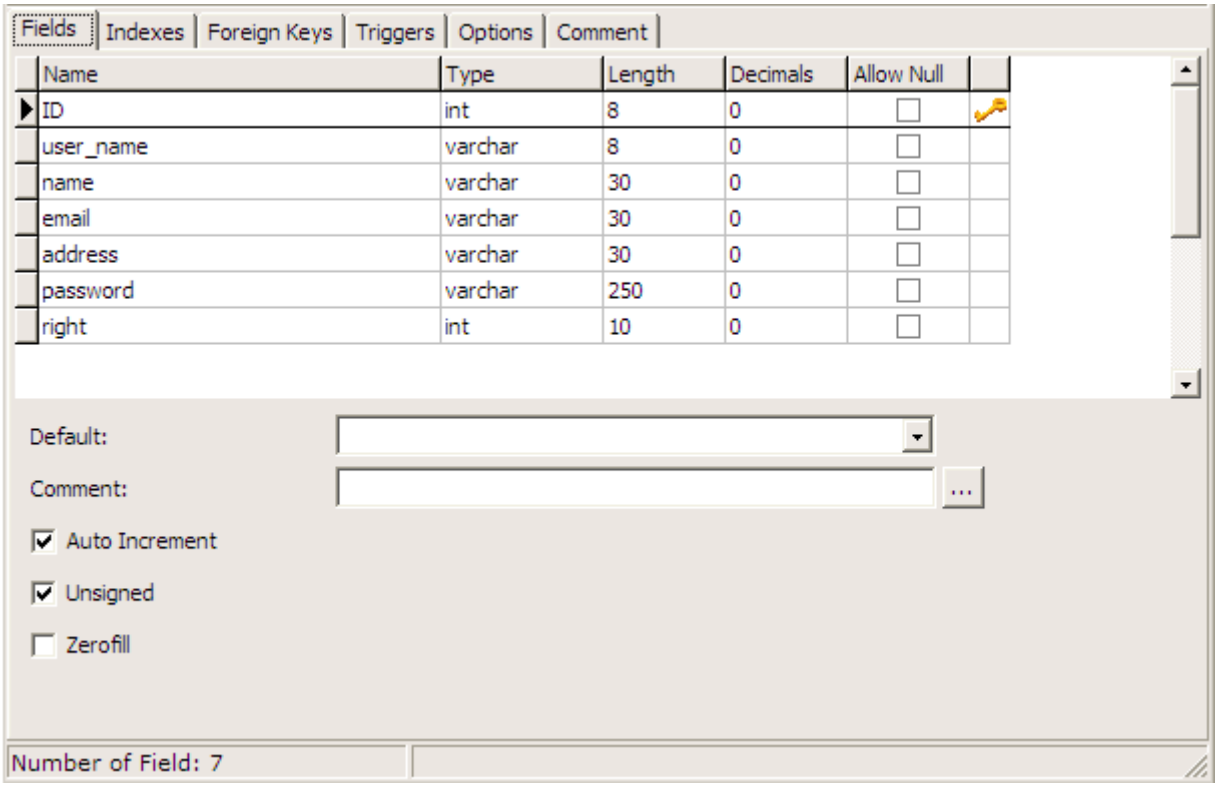
6.3.2. Tervezés újrafelhasználással

Az alkalmazásban újrafelhasználáson alapszik az új jármű felvitele modul. Ez a modul már a rendelkezésemre ált, csak arra volt szükségem, hogy átalakítsam annak érdekében, hogy a megfelelő igényeknek eleget tegyen.

6.3.3. Adatbázis tervezése

Ennek a fejezetnek a szerves részét képezi az adatbázis megtervezése. Mivel egy nem túl nagytömegű adatokkal dolgozó rendszerről van szó, így az adatbázisunk viszonylag egyszerű lesz. Összesen négy táblára lesz szükségünk, melyek a következőek:

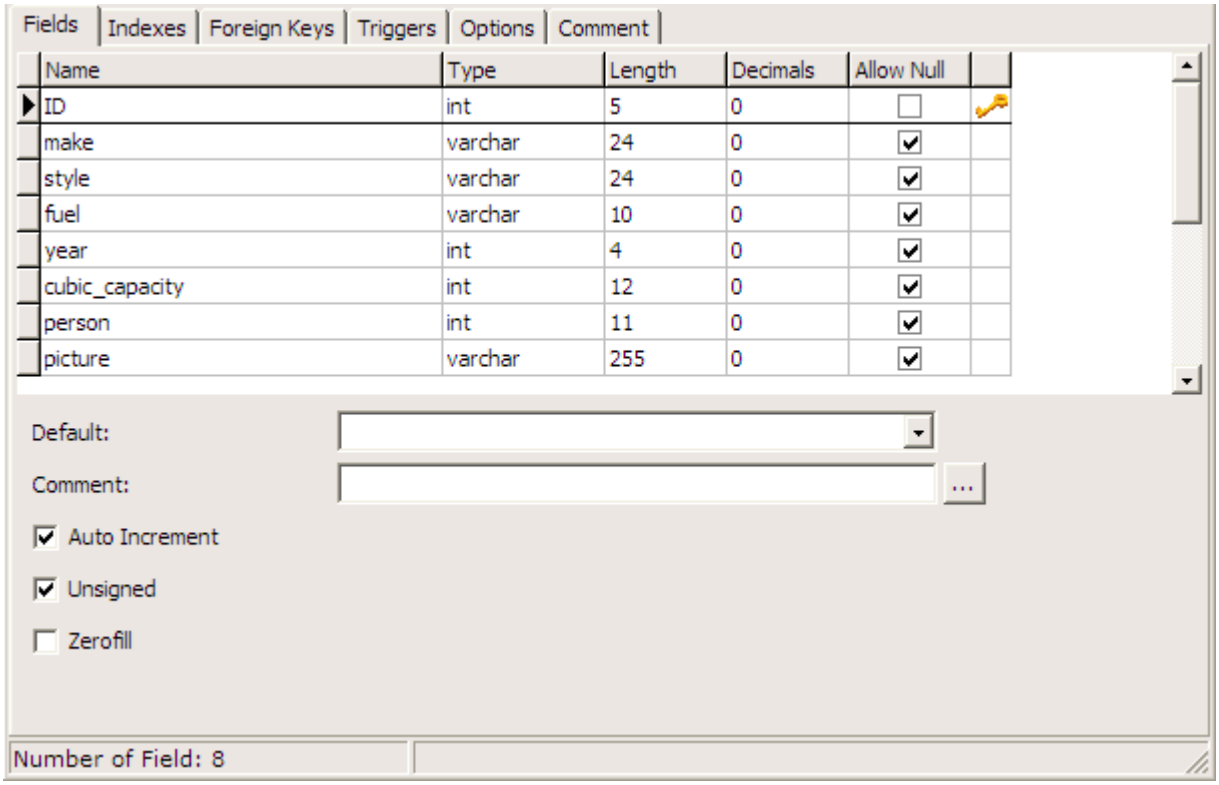
–felhasználó tábla, mely a rendszer felhasználóinak nyilvántartásához szükséges



A mezők:

- ID: a felhasználó egyedi azonosítója, valamint a tábla elsődleges kulcsa. A rendelés táblában erre a mezőre hivatkozva tartjuk nyílván, hogy az egyes járművet melyik felhasználó kívánja kölcsönözni.
- user_name: felhasználói név
- name: a felhasználó teljes neve
- email: a felhasználó e-mail címe
- address: a felhasználó postai címe
- password: a felhasználó rendszerbelépési jelszava
- right: a felhasználó jogosultsági szintje. Ez a mező azért vált szükségszerűvé, mert meg kell különbözetnünk egymástól három felhasználói csoportot. Ezek a csoportok: a nem regisztrált ügyfél, aki csak böngészési joggal rendelkezik; a regisztrált ügyfél, aki a böngészési jogon túl kölcsönzési igényeket jegyezhet az adatbázisba; valamint az adminisztrátor, akinek lehetősége nyílik az egyes felhasználó jogosultsági szintjének módosítására, új járművek adatbázisban való tárolására, illetve az egyes kölcsönzések kezelésére.

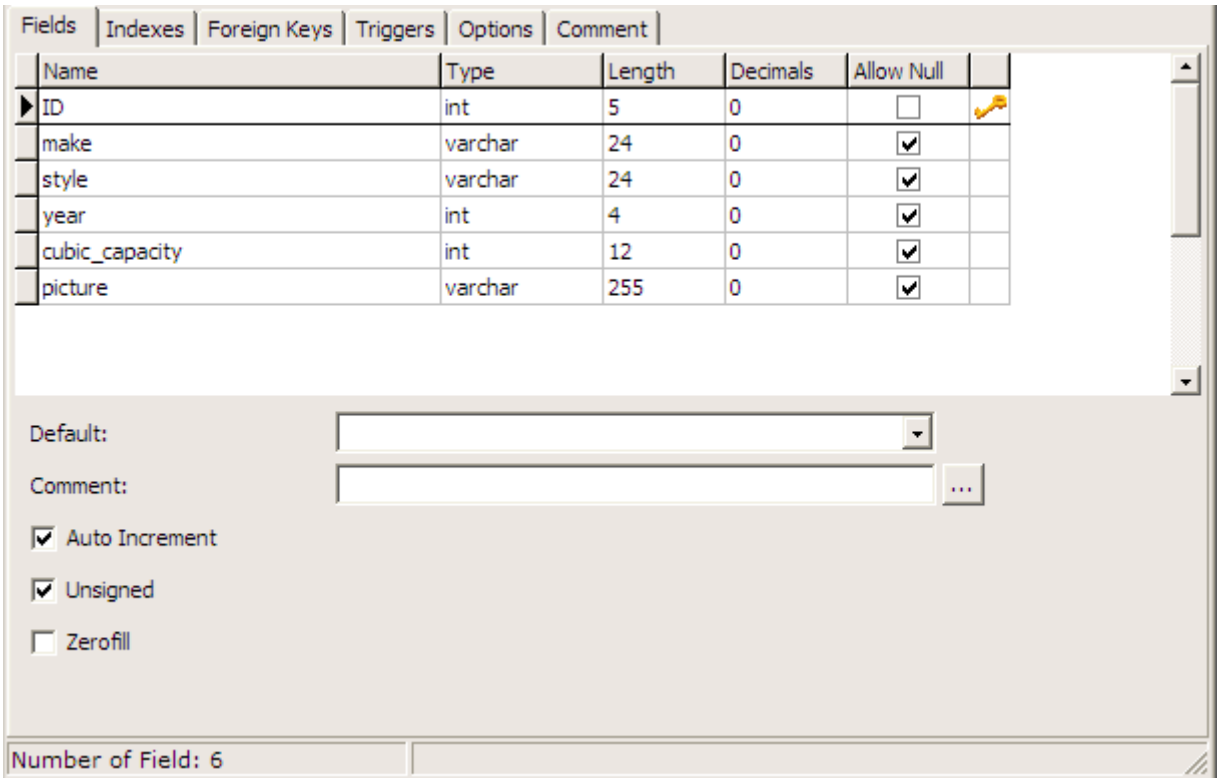
–autó: mely a veterán autókat tárolja



A mezők:

- ID: az autó egyedi azonosítója, valamint a tábla elsődleges kulcsa. A rendelés táblában erre a mezőre hivatkozva tartjuk nyílván az egyes autókra leadott rendeléseket.
- make: az autó gyártmányának neve
- style: az autó pontos típusa
- fuel: az autó üzemanyagtípusa
- year: az autó gyártási éve
- cubic_capacity: az autó hengerűrtartalma
- person: szállítható személyek száma
- picture: az autó fényképének az elérési útja

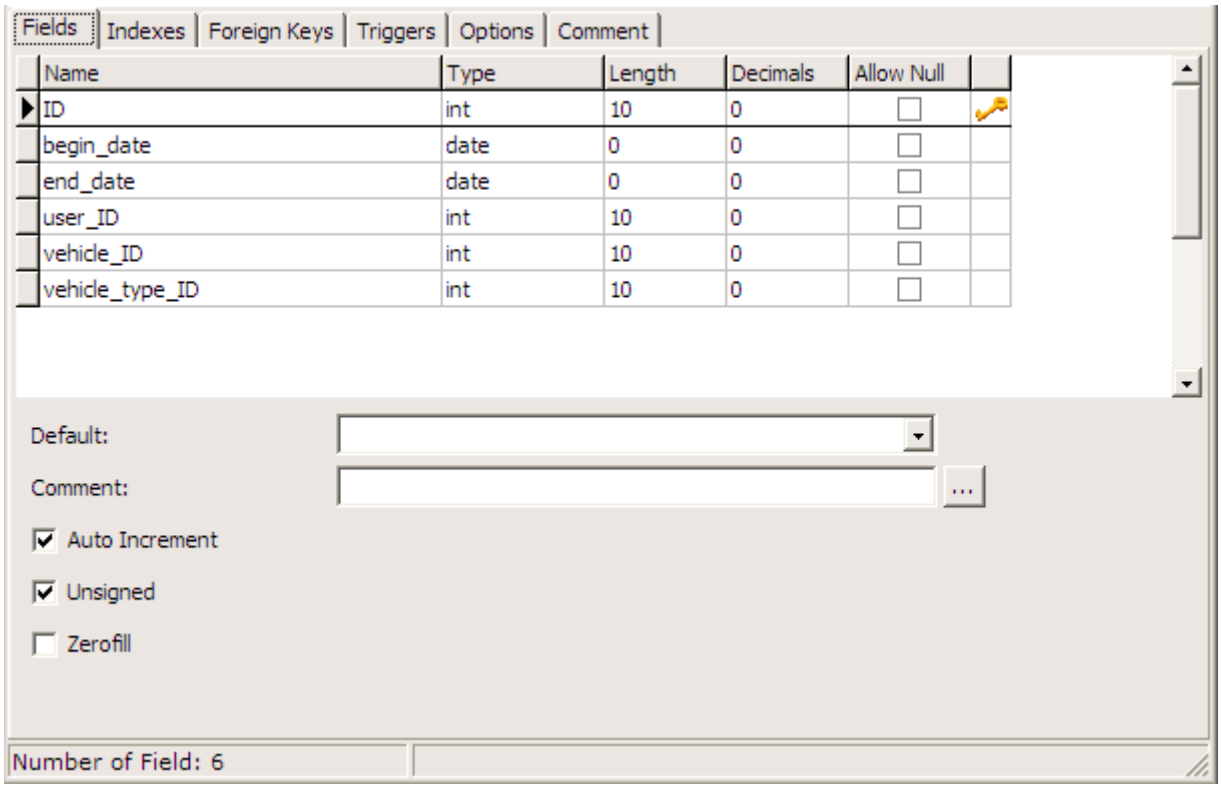
–motor: mely a veterán motorokat tartalmazza



A mezők:

- ID: a motor egyedi azonosítója, valamint a tábla elsődleges kulcsa. A rendelés táblában erre a mezőre hivatkozva tartjuk nyílván az egyes motorokra leadott rendeléseket.
- make: a motor gyártmányának neve
- style: a motor pontos típusa
- year: a motor gyártási éve
- cubic_capacity: a motor hengerűrtartalma
- picture: a motor fényképének az elérési útja

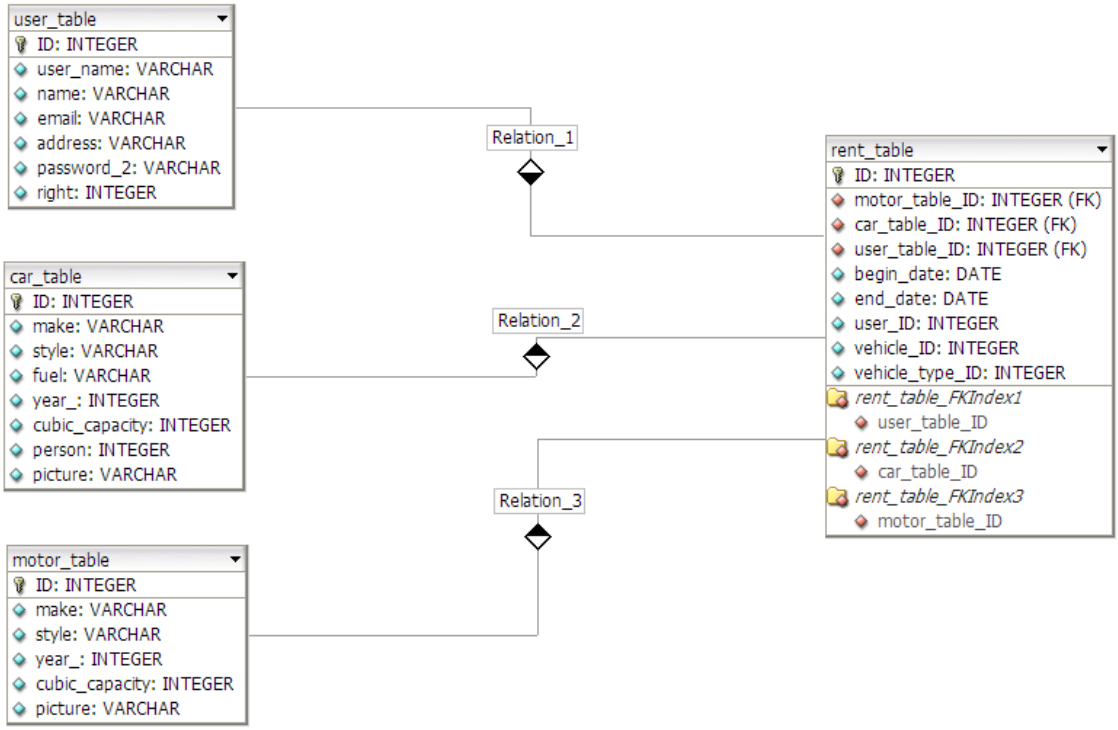
–rendelés: mely az egyes járművekre leadott kölcsönzési igényeket tartja nyílván



A mezők:

- ID: a rendelés egyedi azonosítója, valamint a tábla elsődleges kulcsa.
- begin_date: a kölcsönzési idő kezdete
- end_date: a kölcsönzési idő vége
- user_ID: annak a felhasználónak az ID-ja, aki az adott járműre a kölcsönzési szándékát rögzítette. A felhasználó táblához tartozó külső kulcs.
- vehicle_ID: annak a veterán járműnek az ID-ja, amelyre vonatkozóan kölcsönzési igényt rögzítettek. A járművekhez tartozó külső kulcs.
- vehicle_type_ID: ennek a mezőnek a segítségével tároljuk el, hogy az adott kölcsönzés autóra vagy motorra vonatkozik.

Kapcsolat a táblák között:



Az adatbázisban három kapcsolatot definiáltunk az egyes táblák között. Mindegyik 1:n kapcsolat.

- Relation_1: a felhasználó és a rendelés táblák közötti kapcsolat. Ez a kapcsolat azért szükséges, hogy az egyes felhasználók kölcsönzéseit nyílván tudjuk tartani, és segítségével egy felhasználóhoz hozzá tudjuk rendelni az ő saját kölcsönzéseit.
- Relation_2: az autó és a rendelés táblák közötti kapcsolat.
- Relation_3: a motor és a rendelés táblák közötti kapcsolat.

Az utóbbi két kapcsolat azért szükséges, hogy nyílván tudjuk tartani, hogy milyen veterán járműveket szeretne az adott felhasználó kölcsönözni.

7. Felhasználói felületek

Mindeddig csak a rendszer felépítésének megtervezésével foglalkoztunk, holott egy felhasználó ezekkel a részekkel nem fog közvetlenül találkozni, a rendszer működése a háttérben húzódik, és csak ennek az eredményei fognak megjelenni. A program végrehajtása

során a feldolgozás eredményét valamilyen formában prezentálni kell, tehát szükség van egy felhasználói felületre, amelyen keresztül a felhasználó érintkezni tud az alkalmazással, annak utasításokat tud adni, és amelyről tudja olvasni a kiadott utasítások eredményét. Tehát mivel a felhasználó a felületen keresztül kapcsolódik az alkalmazáshoz, a világ legjobb alkalmazása is tönkremegy, ha a felhasználói felület használhatatlan, ha azon keresztül nem tudjuk használni, mert könnyezik a szemünk, akkor innentől kezdve a teljes alkalmazás is használhatatlan. Az szép dolog, hogy például kiválasztunk egy színt magunknak, és az lesz a kedvencünk, a felhasználói felületen az dominál, csak épp lehet, hogy a felhasználókat nagyon zavarja.

7.1. A felhasználói felület tervezése

- felhasználói jártasság: nem szabad egy felületet azért adni egy felhasználónak, mert azt könnyű implementálni, hanem a felhasználó számára megszokott terminológiát kell használni, megszokott és az alkalmazás környezetével összhangban lévő objektumoknak kell megjelenni a felületen. Ismerős, megszokott környezetbe kell vinni a felhasználót akkor is, ha ez új alkalmazás számára.
- konzisztencia: az alkalmazás felhasználói felületeinek elemei ugyanúgy illik, hogy működjenek minden esetben. Például: ha egy menüben az utolsó a kilépés menüpont, akkor minden menüben utolsó legyen a kilépés; ha azt szokjuk meg egy felhasználói felületen, hogy kétszer kell kattintani, ahhoz, hogy elfogadásra kerüljön, akkor kétszer kelljen kattintani mindenhol. Ez egy úgynevezett alacsonyszintű konzisztencia, tehát amikor a felhasználói felület elemei ugyanúgy működnek. Magasabb szinten teljes konzisztenciát nem lehet elérni, nem lehet azt megtenni, hogy minden mindenütt ugyanúgy működjön, de alapelemek esetén mindig ugyanúgy kell, hogy működjenek.
- minimális meglepetés elve: a felhasználók nem először találkoznak alkalmazásokkal, megszoktak bizonyos szabványműködéseket. Ha valami nem a megszokottként történik, a felhasználó összezavarodik.
- felhasználói támogatás: egy súgórendszernek ott kell lennie a felületen, valamint ha esetlegesen valami kivétel váltódna ki az alkalmazásban, akkor a felhasználónak érthető, világos hibaüzenetet kell küldeni.
- felhasználói sokféleség elve: arra kell felkészülni a felhasználói felületnél, hogy különböző felhasználói szokások vannak. Van, aki az egeret szereti, van, aki a

gyorsbillentyűket, van, aki az ablakos környezet szereti, van, aki a parancssorosat stb. A felhasználói felületnek ki kell szolgálnia minden egyes felhasználót.

7.2. A felhasználói felületek ergonómiája

Az utóbbi időben egyre hangsúlyosabbá vált a felhasználói felületek ergonómiája. Mennyire esztétikusan van a felhasználói felület megtervezve. A megjelenítés, interakciók, színek, egyszerű kezelhetőség és tanulhatóság kérdéskörét vizsgálja.

A felhasználói felület mögött ott ül a végfelhasználó és dolgozni szeretne vele. Az ergonómia annyit jelent, hogy a felhasználói felület használható, tehát a felhasználó olyan képernyőképet kap, amelyet átlát, napi munkájához, tevékenységeihez megfelelő. A képernyőn elhelyezett kitöltendő űrlapelemek elhelyezése fontos, nem jó, ha sok szemmozgással lehet kideríteni, hogy ott mi is akar lenni.

A színek lehetnek a legerőteljesebb zavarótényezők a felhasználó szemében. Itt alkalmazandó a színhasználati ökölszabály, hogy konzervatív módon használjuk a színeket:

- egy felhasználói felületen megjelenő színek száma kevés (legfeljebb 5) legyen, 7-nél több semmiképp;
- a színeket funkcionálisan kell használni és nem önmagukért; a színek használhatók valamilyen állapotváltozás jelzésére (pl. folyamat állapotjelző színe a folyamat közben adott színű, folyamat befejeztével átvált)
- nagyon lényeges funkció kiemelésére (pl. kivételes esetet kezelő funkcióhoz rendelhetünk színt). Bizonyos szakmákban, kulturális környezetekben bizonyos jelentésekhez bizonyos színek társultak. Ezeket a kialakult színkonvenciókat kell használni.
- nagyon meg kell gondolni a színek párosítását, bizonyos színkombinációkat az ember szeme nehezen visel el, bizonyos színkombinációk pedig kellemesek. Például a piros és kék színek fárasztóak az emberi szemnek.
- bizonyos színek bizonyos érzelmi állapotokat váltanak ki az emberből
    - a megnyugtató színek a pasztellszínek
    - a mi kultúránkban a vörös szín veszélyt vagy tiltást jelent

- a kék egy hideg, rideg szín

– különböző monitortípusok különbözőképpen jelenítik meg a színeket (különösképpen a vörös és a barna színeket), ezért vezették be azt a 256-elemű standard színskálát, amelynek színeit elviekben minden monitornak ugyanúgy kellene megjelenítenie.

A felhasználói felület tervezése nem informatikus feladat.

A felhasználói felületekhez hozzátartozik a felhasználói támogatás, elvárjuk, hogy legyen egy online súgórendszer, továbbá manapság elvárjuk, hogy a felhasználói támogatásba beletartozzon az, hogy a felhasználói felületen normális rendszerüzenetek jelenjenek meg egy-egy felhasználói interakcióra.

Mind a súgórendszer, mind a rendszerüzenetek szövegesek, ezeknek a rendszereknek megvannak a maguk kialakult ökölszabályai, szabványai. Ezeknek – ugyanúgy, mint általánosságban a teljes felhasználói felületnek – figyelembe kell venniük sok mindent a felhasználók vonatkozásában.

Az online súgó használatát nagyon befolyásolja, hogy a felhasználónak mekkora tapasztalata van a rendszerrel kapcsolatban, amelyet a felhasználói felületen keresztül ér el: minél tapasztaltabb, annál rövidebb, lényegre törőbb szövegeket szeret olvasni, minél tapasztalatlanabb, annál részletesebb szövegeket vár el. Egy súgórendszernél mindkét véglet igényét kielégíteni lehetetlenség, de valamilyen módon erre gondolni kell.

Elsősorban a rendszerüzeneteknek, de a súgónak is segítőkésznek, udvariasnak, barátságosnak kell lennie, nem lekezelőnek, nem gúnyosnak. A rendszerüzeneteknek mindenképpen informatívnak, és a felhasználó számára értelmezhetőnek kell lenniük, és megjelenésüket ne kísérje hang. Valamint az adott szakterület nyelvén legyen megfogalmazva. Egy felhasználói felületnek el kell rejtenie az információs rendszer informatikai jellegű (operációs rendszer, adatbázis-kezelő rendszer, Java) hibaüzeneteit.

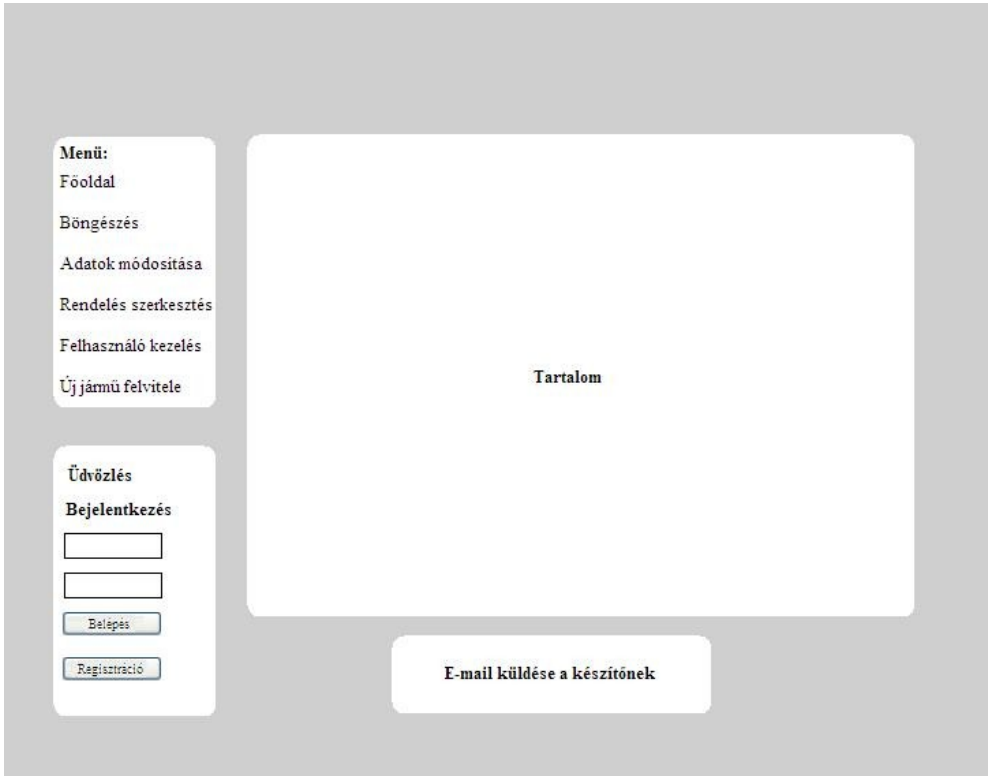
7.3. Veterán Autó- és Motorkölcsönző

7.3.1. Felhasználói felület

Mindezen kritériumokat figyelembe véve próbáltam elkészíteni a példaalkalmazást. Ügyeltem arra, hogy a felhasználó hosszabb idejű böngészés után is szívesen nézegesse a járműveket, kedvére válogathasson közülük, tehát úgynevezett szembarát színeket próbáltam meg használni, mely nem erőlteti meg a szemet, mégis elég figyelem-, és érdeklődésfelkeltő.

Ügyeltem a megszokás elvére, vagyis a menürendszer minden egyes oldalon ugyanolya, tehát ha egy felhasználó már megszokta az alkalmazást, akkor teljes magabiztossággal használhatja azt.

7.3.2. Felületterv



Figyeltem arra, hogy ha valamilyen oknál fogva kivétel adódna az alkalmazás futása közben, akkor a felhasználónak érthető hibaüzenetet küldjön a rendszer. Ilyen például a kölcsönzési igény leadásakor a nem megfelelő dátumintervallum választása, amely esetén a rendszer szól a felhasználónak, hogy nem megfelelő dátumintervallumot adott meg a rendelés közben.

Hasonlóan hibaüzenetet eredményez az az eset, amikor a felhasználó egy járművet szeretne kölcsönözni, de az a jármű az adott időszakban nem állhat a rendelkezésére, mert azt már egy másik felhasználó birtokolja az adott időben. Ekkor a rendszer értesíti a felhasználót arról, hogy az adott járművet az adott időszakban vagy annak egy részében már másik felhasználó birtokolja.

8. Verifikáció és validáció

A követelményeket összegyűjtöttük, elemeztük, eldöntöttük, hogy milyen modell alapján fejlesztünk, megterveztük az architektúrát és a felhasználói felületet, implementáltuk a tervet. Ezután következik a verifikáció és validáció lépése.

A verifikáció azt jelenti, hogy ellenőrizzük, jól készítettük-e el a szoftvert, azt döntjük el, hogy a szoftver megfelel-e a specifikációnak, valamint kielégíti-e a követelményeket.

A validáció ennél egy bővebb dolog, ugyanis a validációnál azt vizsgáljuk, hogy a felhasználó igényeinek megfelelő szoftvert készítettük-e el. Előfordulhat ugyanis, hogy bár a követelményspecifikációt teljes egészében megvalósítottuk, lehet, hogy ez a követelmény-specifikáció hibás, és ekkor nem a felhasználó számára szükséges szoftvert készítettük el.

Tehát a validált szoftver olyan verifikált szoftver, mely a felhasználó tényleges igényeit szolgálja.

Két nagy módszer alakult ki a verifikációra és validációra:

- átvizsgálások módszere,
- tesztelés.

Az átvizsgálás egy statikus módszer, úgy végezzük a lépéseket, hogy közben nem futtatjuk a programot. Megnézzük a szoftver forráskódját, hogy a specifikációnak megfelel-e. Bármilyen dokumentációt, bármilyen részterméket és a programkódot magát át tudjuk vizsgálni. Ez az átvizsgálás általában manuális, emberi tevékenység. Átvizsgálás során ellenőrizzük a dokumentációk, az ábrák és a kód konzisztenciáját.

A tesztelés egy dinamikus módszer, úgy végezzük, hogy futtatjuk a programot speciális, megtervezett tesztadatokkal, és a kapott eredményeket összehasonlítjuk az elvárt eredményekkel. Az átvizsgálás csak verifikációra jó, a tesztelés viszont verifikációra és validációra is jó.

A verifikáció és validáció folyamata sosem mondja azt, hogy a szoftver 100%-os, legfeljebb annyit tud mondani, hogy a szoftver a követelményeknek lényegében, többnyire megfelel, a szoftver a felhasználónak valószínűleg jó lesz. A tesztelés nem alkalmas arra, hogy a szoftverről azt állítsuk, hogy helyes. Nagy kérdés, hogy mikor mondhatjuk azt, hogy a szoftver megfelelő.

8.1. Elfogadási szint

Ha nem minden funkció működik 100%-osan, van olyan kevésbé lényeges, kevésbé gyakran használt, kevésbé általános funkció, amely nem tökéletesen működik, attól maga a szoftver valóban jól használható, ha tudjuk, hogy mi nem működik benne. Tehát ha néhány százaléknyi funkció nem működik, az elviselhető általános szoftvereknél, azonban nyilvánvalóan egy biztonsági szoftvernél a néhány százalék már nem megengedhető. Attól függ, hogy milyen jellegű, milyen területen használt szoftverről van szó.

Az elfogadhatóságnál alapprobléma a piac és a piaci környezet. Ha szoftvert fejlesztünk, és ez elkészül, akkor a fejlesztőnek kell tesztelnie, ez az úgynevezett alfa-teszt.

Ekkor előáll egy alfa-verzió, ez kerül ki először a fejlesztőcsapat kezéből. Ezek után a rendszert kiadják a fejlesztőtől független embereknek béta-tesztelésre, ezt lehetőleg felhasználói körülmények között kell végrehajtani. A béta-tesztelés elsősorban hiányosságtesztelés, és csak másodsorban statisztikai tesztelés. Ezen tesztelésnél feltárt hiányosságok megszüntetése után áll elő a béta-verzió.

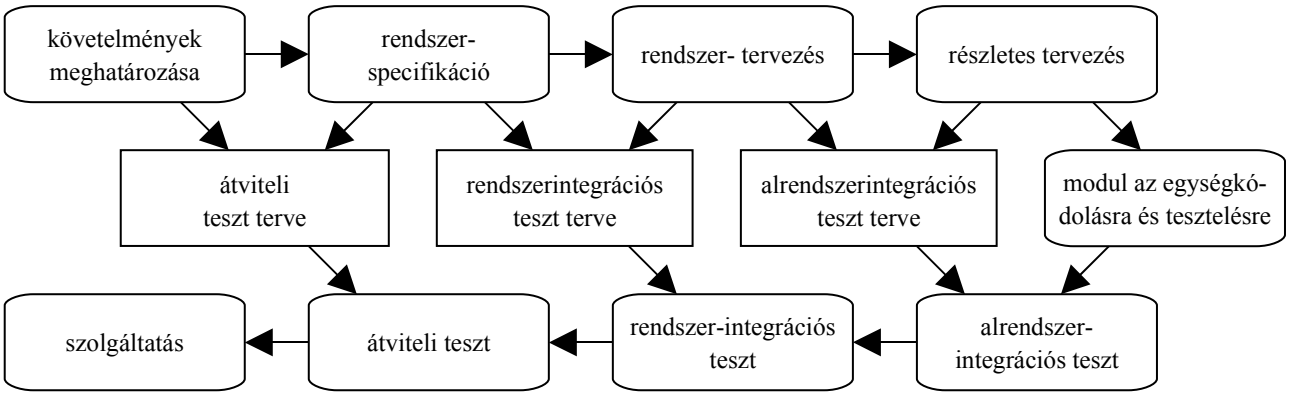
A tesztelés kideríti

- a hiányosságokat a szoftverben,
- a teljesítményproblémákat,
- a specifikációnak nem megfelelést.

Ezután be kell határolni a hibákat. Ez nem a tesztelés feladata. Tehát el kell különíteni a tesztelést, mint verifikációs és validációs lépést, és a hibák kiderítését. Utóbbi a debugolás, belövés. Ekkor tudjuk, hogy rossz a szoftver, meg kell keresni, hogy hol a hiba, ezután kijavítani. A hiba hatása általában később jelentkezik.

Kiderítettük, behatároltuk, kijavítottuk a hibákat, ezután újratesztelünk. Ezt a tesztelést hívjuk regressziós tesztelésnek. Ugyanis lehetséges, hogy ha kijavítunk egy hibát, az három újabbat eredményez a rendszeren belül. A regressziós tesztelés alatt azt értjük, hogy ugyanazokat a teszteket futtatjuk le, mint amiket kezdetben, legalábbis elméletben.

A szoftverfejlesztés legkényesebb, legköltségesebb része a verifikáció és validáció. Ma nagy rendszereknél a verifikáció és validáció a projektköltségnek közel a fele (40-50%). Ennek megfelelően a verifikáció és validáció lépését is tervezni kell.



A tesztelés tervezésénél a következőket kell megtervezni:

- először a teljes tesztelési folyamatot tervezni kell;
- a tesztelést úgy kell megtervezni, hogy a követelményeket egyenként lehessen tesztelni;
- meg kell nevezni a szoftver azon elemeit, amelyeket az adott lépésben tesztelni kell;
- a tesztelést ütemezni kell;
- a tesztelést természetesen dokumentálni kell;
- meg kell tervezni a tesztelő eszközöket amennyiben szükségesek, és hogy ezek hol szükségesek.

8.2. Átvizsgálás fogalma

Az átvizsgálás a teljes fejlesztési folyamat minden egyes lépésében alkalmazható. A dokumentumokat, mint résztermékeket tudjuk átvizsgálni, valamint át tudjuk vizsgálni a szoftvert. Ezt a szoftverátvizsgálást hívja a szakmai szleng száraztesztnek. Az átvizsgálásra gyakorlatilag különböző módszerek alakultak ki a 70-es évektől kezdve, különösen az IBM dolgozott ki ilyen módszereket, átvizsgálási technikákat.

A szoftverátvizsgáláshoz nem kell futtatni a programot. Ez egy olcsó és hatékony módszer a hiányosságok tesztelésére. Tehát a szárazteszt tipikusan egy hiányosságtesztelés, egy verifikációs technika. Nyilvánvalóan a forráskódból ránézésre elég nehéz megmondani a válaszidőt és egyéb statisztikai jellemzőket. Ugyanakkor hiányosságtesztelésre meglehetősen alkalmas mivel:

- az átvizsgálás folyamán egyszerre több hiányosságra is fény derülhet, tehát a hiányosságok csoportját tudjuk az átvizsgálás során felderíteni.
- ha futtatjuk a programot (dinamikus tesztet hajtunk végre), akkor a hiányosságtesztelésnek van egy olyan kellemetlen mellékhatása, hogy a hiányosságok valószínűleg az adatok leromlását eredményezik, korrodálódnak az adatok a tesztelés folyamán és akkor más adatot tesztelünk, tehát vissza kell állítani az eredeti adatokat.

– az igazán jó tesztelők azok, akik az adott területet, nyelvet, technikákat, platformot jól ismerik.

– A programátvizsgálás folyamán felismerhetők, felismerendők a következők:
– adatkezelési hibák
    - változónak van-e kezdőértéke?
    - tömbindex hivatkozások méreten belül vannak-e?
    - elágaztató utasítások feltételei megfelelőek-e?
    - literálok megfelelő alakúak-e?
– vezérlési hibák
    - minden ciklus minden esetben véget ér?(végtelen ciklusok)
    - elágaztató utasítással minden rendben az adott nyelvnél? (break, default ág, csellengő ELSE)
– I/O hibák
    - az input adatok mindegyikére szükség van? felhasználja a program?
    - ki vannak védve az input-hibák? minden inputra működik a program?
    - output-változóknak adtunk értéket?
– interfész problémák
    - alprogramok, metódusok paraméterkiértékelésénél, -átadásánál minden rendben van? (típus, sorrend, szám)
– tárkezelési hibák különös tekintettel a dinamikus szerkezetekre
    - mutatók, NULL-mutatók használata hivatkozásra
    - tárfoglalás
– kivételkezelés
    - minden kivétel le van kezelve?
    - ott van lekezelve, ahol kell?

A programátvizsgálásokhoz léteznek nyelvspecifikus automatikus eszközök, megoldható az automatikus átvizsgálás (automatikus statikus tesztelés). Ezek a statikus tesztelők a program szövege alapján megpróbálják felderíteni a fenti kategóriájú problémákat. A statikus elemző a fordítóprogrammal együtt, a mellett működik. A fenti problémák felderítését célozza. Amit kézzel nem lehetne megoldani, viszont a statikus elemző képes rá,

az az útvonalelemzés. Azonosítja a programon belül az összes lehetséges végrehajtási útvonalat, és elemzi őket. Akár végtelen sok is lehet, ezért a végtelen ciklus felderítésére is alkalmas. Nyilvánvalóan nyelve válogatja, hogy mennyire könnyű vagy nehéz egy ilyen automatikus eszközt megírni. Szigorúan típusos nyelvek esetén bizonyos dolgokat sokkal könnyebb felfedezni. Vannak olyan dolgok, amelyeket bármilyen nyelvnél könnyen lehet tesztelni, például paraméterek számossága; szigorúan típusos nyelvnél könnyű ellenőrizni, hogy a formális és aktuális paraméterek típusa megegyezik-e. Azt viszont nem lehet kideríteni automatikus statikus elemzővel, hogy bár a típus jó, teljesen mást adtunk át. Ez szemmel könnyebben észrevehető.

9. Összegzés

A webes alkalmazásfejlesztés napjainkban nélkülözhetetlen. Célunk, hogy a felhasználók igényeit minél nagyobb mértékben teljesítsük. A fejlesztés során szükségszerű, hogy kapcsolatban maradjuk a felhasználókkal, hogy a legmegfelelőbb alkalmazást tudjuk elkészíteni számukra.

Diplomamunkám elkészítése során megpróbáltam a lehető legjobban kifejteni a webes alkalmazásfejlesztés teljes életciklusát. Minden fejezet végén az elméleti tudást kivetítettem a gyakorlati alkalmazásra egy példa alapján. Mivel programtervező matematikus vagyok, így a hangsúlyt a tervezésre helyeztem leginkább. Nagyon fontos, hogy a felhasználóktól megtudjuk a pontos követelményeket, hiszen ezek hiányában sohasem tudjuk elkészíteni a megfelelő alkalmazást, sohasem lesznek elégedettek a felhasználók, így elveszíthetjük őket. Nagyon fontos, hogy minél nagyobb mértékben megfeleljen egy webes alkalmazás a felhasználói igényeknek, és ezáltal célközösségünk elégedetten hagyja majd el az oldalunkat, és ismételten visszatér később. Ezzel sokat nyerünk mind gazdaságilag, mind elismerésileg, mind pedig a felhasználók számát illetőleg.

Diplomamunkám igazából akkor érte el a célját, mikor sikeresen befejeztem az első önálló alkalmazásom fejlesztését és tesztelését, és nyugtáztam, hogy az a legtöbb felhasználói kritériumnak megfelel. Elkészítése során böngészéseim alkalmával minden egyes újonnan megtekintett oldalnál elgondolkoztam, vajon én azt várom-e el az oldaltól, amit látok, és amit

tapasztalok. Ez nagyban segítette a további munkámat. Egyre jobban beleástam magam a webes alkalmazásfejlesztés témakörébe, egyre többet olvastam, és ami a legfontosabb egyre többet tanultam meg róla.

Úgy gondolom, hogy a diplomamunkám elején megfogalmazott célokat megfelelően teljesítettem. A diplomamunkám elkészítése során érlelődött meg bennem a gondolat, hogy a jövőben is ehhez hasonló kihívásokat szeretnék keresni, és ami legfontosabb meg is szeretném őket valósítani.

10. Irodalomjegyzék

Gál Tibor: Web programozás

Ian Sommerville: Szoftverrendszerek fejlesztése

Matt Zandstra: tanuljuk meg a PHP5 használatát 24 óra alatt

Julie C. Meloni: A PHP, a MySQL és az Apache használata

http://www.w3.org/

http://www.w3schools.com/CSS/pr_class_visibility.asp

http://weblabor.hu/cikkek/designbolkeszoldal

http://www.tankonyvtar.hu/informatika/objektum-orientalt-080905-184

11. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani:

- Dr. Kuki Attilának, amiért vállalta a diplomamunkám témavezetését, és javaslataival nagyban segítette a munkámat.

- Szüleimnek és testvéremnek, akik mindvégig biztattak tanulmányaim során, és biztatnak a mai napig. Akik mindig is szeretettel és teljes odafigyeléssel hallgatták

végig a néha szinte érthetetlen beszámolóimat a programozás és az egyetemi élet területéről.

- Nagy Péternek, a barátomnak, aki rengeteg segítséget nyújtott számomra az egyetemi tanulmányaim során, és sok ötletet adott a diplomamunkám elkészítéséhez.

- Bíró Gábornak, aki létrehozta a Veterán Kulturális és Sport Egyesületet, és ezzel ötletet adott a témaválasztáshoz.