DIPLOMAMUNKA

Tarcsa Bálint Dávid

Debrecen

2009

Debreceni Egyetem

Informatikai Kar

WEBES ALKALMAZÁSFEJLESZTÉS SZABADON VÁLASZTOTT TÉMÁBAN

Témavezető:

Dr. Kuki Attila

egyetemi adjunktus

Készítette:

Tarcsa Bálint Dávid

programtervező matematikus

Debrecen

2009

Tartalom

1. Bevezetés 5

2. A szoftverfejlesztés folyamata, modelljei 7

2.1. A vízesésmodell 8

2.2. Evolúciós modell 10

2.3. Formális modell 11

2.4. Komponensalapú modell 12

2.5. Iteratív modell 13

2.5.1. Inkrementális fejlesztés 13

2.5.2. Spirális fejlesztés 14

2.6. A választott fejlesztési modell 15

3. Követelmények meghatározása 16

3.1. Elképzelés (Vízió) 16

3.1.1. A projekt esetén 17

3.2. Megvalósíthatósági tanulmány 17

3.2.1. A projekt esetén 17

3.3. Követelmények feltárása és elemzése 18

3.3.1. A projekt esetén 19

3.4. Fogalomszótár 20

4. Fejlesztői környezet, felhasznált technológiák 21

4.1. IIS 21

4.2. MySQL 21

4.3. HTML 23

4.4. PHP 23

4.5. JavaScript 25

4.6. CSS 25
5. Tervezés 26
5.1. Adatbázis tervezése 26
5.2. Az alkalmazás alrendszerei, moduljai 33
5.2.1. Normál oldal 34
5.2.2. Adminisztrátor oldal 35
5.3. Biztonság 35
5.3.1. Bemenet ellenőrzése 36
5.3.2. Ember vagy gép 37
5.4. Felhasználói felületek 38
5.5. Újrafelhasználás 39
6. Implementáció és egységteszt 40
7. Integráció és rendszerteszt 41
8. Működtetés és karbantartás 42
9. Összefoglalás 43
10. Irodalomjegyzék 45
11. Köszönetnyilvánítás 46

1. Bevezetés

Bár 30-40 évvel ezelőtt aligha gondolták volna, de napjainkra a számítástechnika az élet szinte minden területén megjelent, hogy segítse az embert a hatékonyabb munkavégzésben, tanulásban, információszerzésben, gyógyításban, szórakozásban és még sorolhatnánk sokáig a példákat. Akkoriban az volt az általános vélekedés, hogy a számítástechnika megmarad a kormányok, kutatóintézetek és nagycégek „játékszerének". Mondhatjuk azt, hogy szerencsére nem így történt, így rengeteg előnyét élvezhetjük az elterjedésének. De ugyanakkor fontos arra is figyelmet fordítani, hogy az előnyök mellett éppúgy lehetnek hátrányai is, ha nem megfelelően használjuk. A számítógép például okozhat függőséget, amely az ember szociális környezetét és természetesen önmagát is rombolja, más függésekhez hasonlóan. Erre oda kell figyelnünk, hogy a számítógép, számítástechnika egy hatékony eszköz és nem pedig cél legyen. Természetesen nem ez utóbbival kívánok foglalkozni a dolgozatomban, hanem azzal, hogy miként lehet a számítástechnikát felhasználni – hasznosan – például egy webes alkalmazás segítségével.

A számítógép önmagában nem sok mindenre jó, hiszen a hasznossága csak a rajta futó szoftverek használatával mutatkozik csak meg. Bármilyen gépről is legyen szó, amely valamilyen automatizált folyamatot képes megvalósítani, mindegyik rendelkezik legalább egy programmal. Ez a program lehet úgynevezett beégetett program (pl. egy automata mosógép programja), vagy telepített program, mint például a hagyományos értelemben vett számítógépek operációs rendszerei (Windows, Linux, stb.). Az előbbieket sok esetben nem lehet frissíteni vagy változtatni.

Ha rétegesen ábrázolnánk a hardver és szoftver együttesét, akkor alulról felfelé indulva a hardver lenne a legalsó, a következő az operációs rendszer réteg. Csak ezek felett jelennének meg a további alkalmazások. Nyilvánvalóan egy webes alkalmazás is az operációs rendszer felett van valahol. Általános, hogy a felhasználó egy böngésző szoftveren keresztül találkozik ezekkel az alkalmazásokkal, az interneten (vagy intraneten) keresztül érik el ezeket a szoftvereket. Innen jön a webes alkalmazás elnevezés is.

A webes alkalmazásfejlesztés napjainkban a szoftverkészítés egyik kiemelkedő ágát képviseli. Ennek oka, hogy a webes felületen elérhető alkalmazások egyre nagyobb

népszerűségre tesznek szert. Számos előnnyel rendelkeznek, amelyek révén mind a használatuk, mind a karbantartásuk egyszerűen megvalósítható.

A webes alkalmazások néhány előnyös tulajdonsága:

- Egyszerű telepítés: a kliens számítógépeken legtöbbször nincs szükség telepítésre és konfigurálásra.
- Az Internet felől éppúgy elérhetővé tehetők, mint a helyi hálózatokból. Így távoli munkavégzés során is használhatók.
- Felhasználóbarát kezelőfelület.
- Egyszerűen megtanulható kezelhetőség.
- Jól ismert web böngésző szoftverek segítségével futtathatók.

Mivel már eddig is foglalkoztam internetes oldalak fejlesztésével, adta magát, hogy egy korábbi projektemet használjam fel a diplomamunkám alapjaként. Egy hangszer-zenemű szaküzlet és szerviz webes megjelenését megvalósító szoftver elkészítését kívánom a dolgozatomban bemutatni.

Körül kívánom járni a szoftverfejlesztés folyamatát az egyes részeknél más-más részletességgel. Leginkább azokkal a dolgokkal akarok foglalkozni, amiket lényegesnek találok egy ilyen alkalmazás elkészítésénél (követelmények, tervezés, biztonsági kérdések, stb.) összevetve az elméleti eljárásokkal is.

2. A szoftverfejlesztés folyamata, modelljei

Az 1960-as évek végére megértették azt, hogy a szoftver is egy termék, melyet azért állítanak elő, hogy szolgáltatásokat lásson el. Határidők, költségek vannak, és minőségi fogalom is kapcsolható hozzá. Viszont vannak sajátos jellemzői, melyek megkülönböztetik a többi terméktől. Mindezeket figyelembe véve ugyanúgy szüksége van egy tervezett gyártási folyamatra -, amely magában foglalja a szoftver teljes életciklusát -, mint a hagyományos kézzel fogható termékek esetében.

A tervezettség szükségességére rávilágít a következő, ’50-es (’60-as) évekből származó statisztika: a programok 45 %-a *soha nem futott rendesen*, 30 %-a *fizettek érte, de nem működött jól*, 20 %-a *csak átdolgozás után működött*, 3 %-a *javítás után futott* és csak a maradék 2 % volt az, amely *azonnal futott,* gond nélkül. Nyilvánvalóan ebben közrejátszott a kor technológiai színvonala is, de ezzel együtt is drasztikusan nagy a hiba aránya.

A szoftverkrízis nyomán létrejött felismerés arra késztette a szakembereket, hogy olyan modelleket dolgozzanak ki, amivel kordában lehet tartani a szoftverfejlesztést pénzügy, határidő és minőség tekintetében is. A szoftverfolyamat tevékenységek és kapcsolódó eredmények láncolata, amelyek a termék előállításához vezetnek. Mivel a folyamat egy szellemi folyamat, amely rengeteg emberi sajátosságot igényelnek, ezért az automatizálás csak korlátozott mértékben jöhet szóba. Nem jelenthetjük ki, hogy létezik egy „ideális” szoftverfolyamat, amely mindenki számára megfelelő. Habár számos különböző szoftverfolyamat létezik, alapvetően fejlesztési folyamat négy nagy részre osztható:

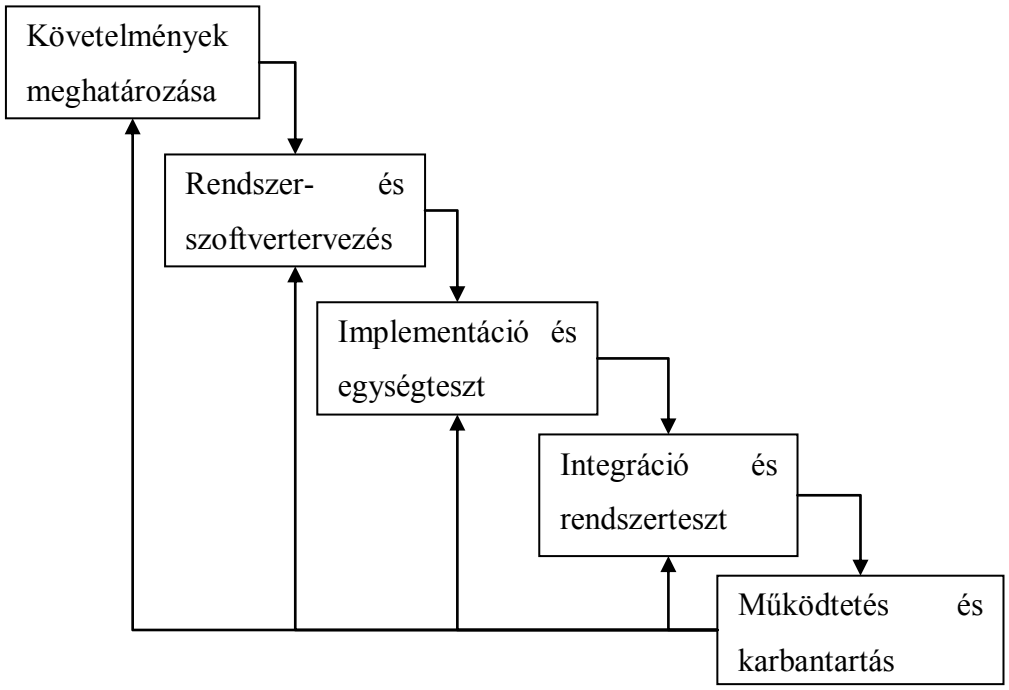
- *Specifikáció.* A szoftver funkcióit, elvárásait és annak megszorításait határozza meg.
- *Tervezés és implementáció.* A specifikációknak megfelelő szoftver előállítása.
- *Validáció.* A szoftver funkcióinak, működésének összevetése az ügyfél kéréseivel, hogy biztosítsuk, a szoftver pontosan azt csinálja, amit az ügyfél elvárt.
- *Szoftverevolúció.* A szoftver továbbfejlesztése az újonnan felmerült igények szerint. Ez nem hibajavítást jelent. E szakasz megléte nem feltétlenül szükséges, de ajánlott, hogy a szoftver úgy készüljön, hogy ez se okozzon különösebb problémát.

A felsorolt szakaszok hatékony kezelésére különböző modellek jöttek létre. A szoftverfolyamat modellje egy absztrakt reprezentáció, amely kiemeli a számunkra fontos

részleteket és ezek alapján közelíti meg a problémát. Az így létrejött modell már áttekinthető, kezelhető. A gyakorlatban az általános modellt konkretizálni kell, meg kell határozni az egyes részek által lefedett feladatokat konkrétan, kiegészítve további szükséges információkkal. A modellekre is igaz, hogy nem lehet azt mondani, hogy valamelyik a sok közül optimális lenne. Ebből kifolyólag hol az egyiket, hol a másikat használják, de gyakran előfordul az is, hogy vegyesen kerülnek felhasználásra. Különösen igaz ez nagy rendszerek esetén, ahol az eltérő alrendszerek más-más modellel kezelhetők a legjobban.

2.1. A vízesésmodell

A szoftverfejlesztés folyamatának első nyilvánosságra hozott modellje más tervezői modellekre épülve jött létre az 1970-es években. Mivel az egyes fázisok lépcsőzetesen kapcsolódnak egymáshoz, a modell vízesésmodellként vált ismertté. Nevét a szemléltető folyamat ábra nevéről kapta:



A modell öt szakasza alapvető fejlesztési tevékenységekre képezhető le:

1. *Követelmények elemzése és meghatározás.* A rendszer szolgáltatásai, megszorításai és céljai a majdani felhasználókkal lezajló megbeszélések alapján alakulnak ki. Ezeket később részletezik, és ezek adják a rendszer-specifikációt.

2. *Rendszer- és szoftvertervezés.* A rendszer tervezési folyamatában választódnak szét a hardver- és szoftverkövetelmények. Itt kell kialakítani a rendszer átfogó architektúráját. A szoftver tervezése az alapvető szoftverrendszer-absztrakciók, illetve a köztük lévő kapcsolatok azonosítását és leírását is magában foglalja.

3. *Implementáció és egységteszt.* Ebben a szakaszban a szoftverterv programok, illetve programegységek összességeként realizálódik. Az egységteszt azt ellenőrzi, hogy minden egyes egység megfelel-e a specifikációjának.

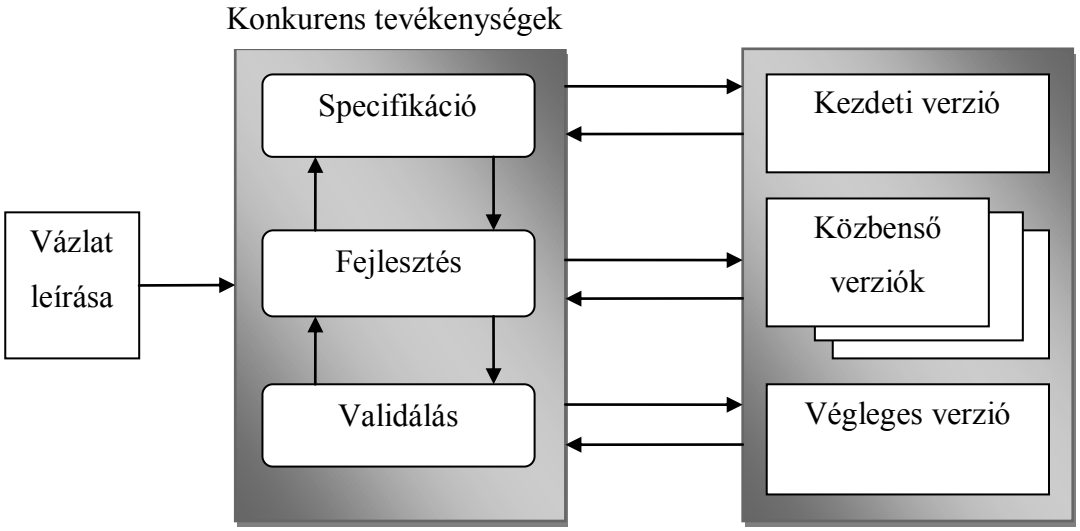
4. *Integráció és rendszerteszt.* Megtörténik a különálló programegységek, illetve programok integrálása és teljes rendszerként való tesztelése, összevetve a megkapott követelményekkel. A tesztelés után a szoftverrendszer átadható az ügyfélnek.

5. *Működtetés és karbantartás.* Többségében ez a szakasz a leghosszabb egy szoftver életében. Megtörtént a telepítés és a rendszer gyakorlati alkalmazása. A karbantartásba beletartozik az olyan hibák kijavítása, amelyek nem derültek ki az életciklus korábbi szakaszaiban, a rendszeregységek implementációjának továbbfejlesztése, valamint a rendszer szolgáltatásainak továbbfejlesztése a felmerülő új követelményeknek megfelelően.

A fázisok lineárisan követik egymást, azaz a következő szakasz addig nem indulhat el, amíg az előző le nem zárult, mégpedig jóváhagyással. A gyakorlatban előfordul, hogy vannak bizonyos átfedések, ugyanis előfordulhat, hogy egy későbbi szakasz világít rá egy problémára és akkor vissza kell térni egy korábbi fázishoz. A fázisok eredménye egy vagy több dokumentum.

A modell leginkább kisméretű szoftverekhez használható a jól szervezhetősége, menedzselhetősége és tervezhetősége miatt. Hátránya viszont, hogy csak akkor alkalmazható jól, ha követelmények teljesen világosak és jól specifikáltak. Nagyméretű szoftverekhez gyakorlatilag használhatatlan. A modell problémáját a projekt szakaszainak különálló részekké történő, nem flexibilis osztása okozza. Ez azt jelenti, hogy már a folyamat korai szakaszaiban állást kell foglalnunk és elköteleznünk magunkat, ami miatt nehéz rugalmasan kezelni az ügyfél által időközben óhajtott változtatásokat. Mindezek ellenére a mai napig igen elterjedt e modell (vagy a modellen alapuló szoftverfolyamat) alkalmazása a tervezők körében, különösen akkor, ha egy nagyobb rendszertervezési projekt részeiben.

2.2. Evolúciós modell



Az evolúciós modell mintegy a vízesésmodell tagadására jött létre. Minél jobban a párhuzamosításra törekszik az egyes lépések tekintetében. Az alapötlet az, hogy először legyen egy kezdeti, gyorsan elkészülő és működtethető szoftverimplementáció minimális funkcionalitással (prototípus), melyet az ügyfél azonnal ki tud próbálni. Majd ezt a prototípust továbbfejlesztve jutunk el a végső verzióig, folyamatosan egyeztetve az ügyféllel. A kezdeti és végső változat között tetszőleges számú közbenső prototípus van. Míg a vízesésmodell esetén az ügyfelet általában csak a követelmények meghatározásánál és a szoftver átadásakor vontuk be a fejlesztésbe, itt folyamatosan szükséges az együttdolgozás. Tehát egy prototípus alapú fejlesztési modellel van dolgunk, amelynek két különböző változata ismert:

1. *Feltáró fejlesztés.* A cél az, hogy a megrendelővel együtt felderítsük a követelményeket, és kialakítsuk a végleges rendszert. A fejlesztést a rendszer már legjobban megértett részével kezdjük. A végleges rendszer úgy alakul ki, hogy folyamatosan építjük be az ügyfél által kért új szolgáltatásokat a már meglévők mellé. A prototípusok egymásra épülnek. Nagyon fontos a világos követelményrendszer.
2. *Eldobható prototípus készítése.* A cél az, hogy megértsük az ügyfelet, hogy milyen szolgáltatásokkal ellátott szoftvert akar. Elég sok a homályos részlet ez esetben, így először ezekre a részekre koncentrálunk és próbálgatással kell rájönnünk arra, hogy mit is akar konkrétan a megrendelő. A név onnan ered, hogy van egy prototípus sorozatunk, ahol a korábbiakat nem használjuk.

A modell hatékonyabb a vízesésmodellnél, ha a rendszer célja az, hogy közvetlenül megfeleljen az ügyfél kívánságainak (akár úgy is, hogy szokatlan megoldásokat alkalmazunk a bejáratottakkal szemben). További haszon adódik azzal, hogy a rendszer-specifikáció folyamatosan fejleszthető, mivel a felhasználók problémái egyre jobban visszaköszönnek a szoftverrendszerben. Itt is elmondható, hogy a legcélravezetőbb kisméretű (500 000 programsorig) szoftverekhez használható a jól szervezhetősége, menedzselhetősége és tervezhetősége miatt.

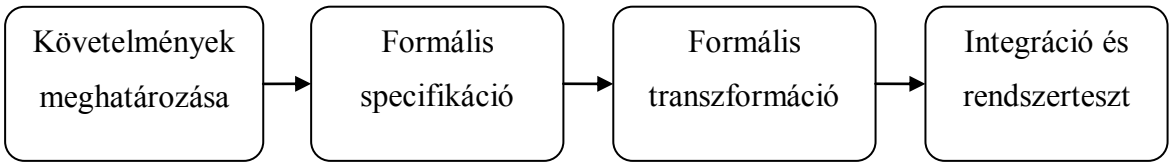
Sajnos ez a folyamatmodell sem hibátlan. Két problémát lehet megemlíteni:

1. *A folyamat nem látható.* A fejlődés mérhetőségéhez szükség van részeredményekre, amelyek folyamatos előállítása minden prototípushoz meglehetősen költséges.
2. *Strukturális anomáliák.* A folyamatos módosítások lerontják a szoftver struktúráját. A szoftver változtatásainak összefésülése pedig egyre nehezebb lesz és rendkívül költséges.

Ezek miatt – többek között - speciális ismeretek, szakemberek és eszközök szükségesek a kezelhetőség érdekében.

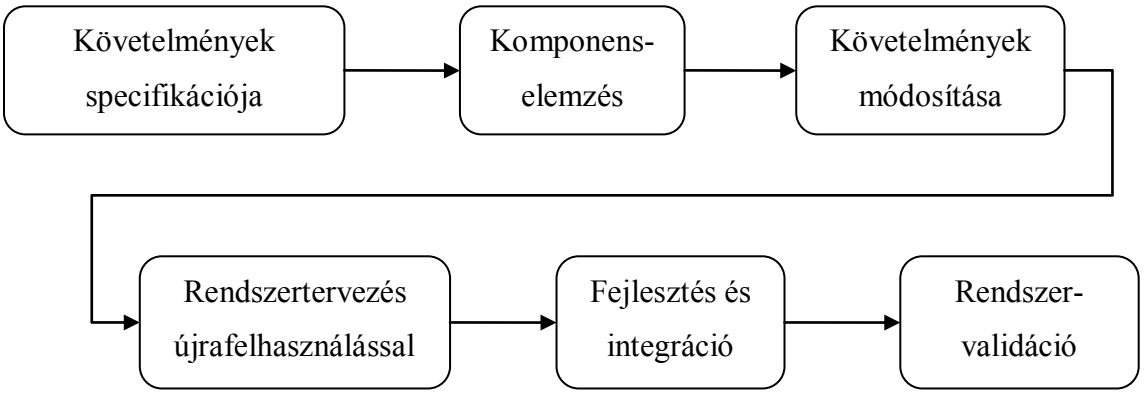
2.3. Formális modell

A Dijkstra-féle struktúra hatására alakult ki, a vízesésmodell absztraktabb változataként. A követelményeket matematikai absztrakció felhasználásával adjuk meg. Ezt a modellt majd matematikai transzformációkkal futtatható kóddá alakítjuk, megőrizve annak konzisztenciáját. Egyik legismertebb példája a Cleanroom folyamat, melyet az IBM-nél fejlesztettek ki. Nagy előnye, hogy a háttérben folyamatosan meghúzódó precíz matematikai formalizmus miatt bizonyítottan helyes az előálló program. Ugyanakkor csak nagyon speciális esetekben használható ez a modell.



2.4. Komponensalapú modell

A név azt takarja, hogy a rendszer fejlesztéséhez használunk már létező komponenseket. A szoftverfolyamatok többségénél megtalálható a szoftverek újrafelhasználása, de ez leginkább a kód újrafelhasználásban merül ki. Itt viszont ezt kiterjesztjük a fejlesztés minden szintjére. Az újrafelhasználás-orientált megközelítési mód nagymértékben az elérhető újrafelhasználható szoftverkomponensekre, illetve azok egységes struktúrába történő integrációjára támaszkodik. Néhány alrendszer a kereskedelemben nagy számban kapható kulcsrakész COTS- (commercial off-the-shelf) rendszer, amelyeket megvásárolhatunk és a rendszerbe integrálhatjuk. Ez általában gazdaságosabb, mint kifejleszteni. A folyamat szemléltetése ábrán:



Az első és utolsó szakasz összehasonlítható más folyamatokkal, de köztük lévő négy már jelentős különbségeket mutatnak ebben a fejlesztésben. Ezek a szakaszok a következők:

1. *Komponenselemezés.* A követelményeket megfelelő módon implementáló komponenseket kell keresni. Általában nincs pontos illeszkedés, ezért a funkcióknak csak egy részét implementálják.
2. *Követelménymódosítás.* A megtalált komponensek információit alapul véve módosítani kell a követelményeket. Ahol a változtatás nem megoldható, ott újra komponenselemzéshez térünk vissza és alternatív megoldást kell találni.
3. *Rendszertervezés újrafelhasználással.* Létező vázat használva vagy újat csinálva megtervezzük a rendszert számításba véve az újrafelhasználandó komponenseket, hogy azok működhessenek. Újrafelhasználható összetevők hiányában új szoftverek is kifejleszthetők.

4. *Fejlesztés és integráció.* Megtörténik a nem megvásárolható komponensek kifejlesztése, és a COTS-rendszerekkel történő integrációja. A rendszer-integráció itt sokkal inkább a fejlesztési folyamat része, mint különálló tevékenység.

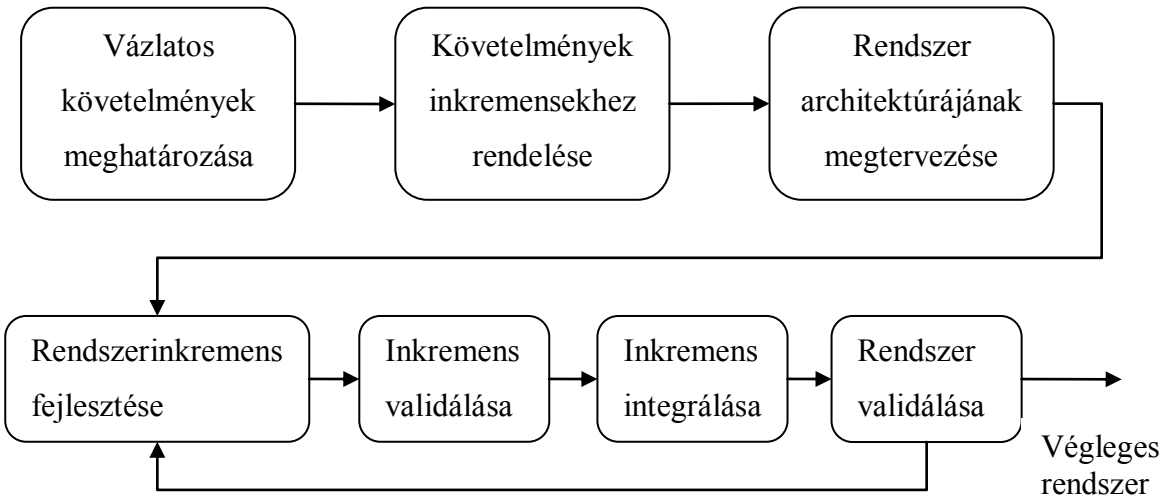
A modell nyilvánvaló előnye az időtakarékosság, ami pozitív kihatással van a fejlesztési, költség és kockázati tényezőkre egyaránt. Viszont ennek az ára az lehet, hogy sokszor kompromisszumokat kell kötni a komponensek által nyújtott szolgáltatások és a felhasználó eredeti kívánságai között. Továbbá meg kell, bizonyosodjunk arról, hogy nincsenek rejtett, nem kívánt dolgok a komponensekben, amelyek kihatással lehetnek a rendszerünk működésére.

2.5. Iteratív modell

A modell lényege, hogy a szoftverfejlesztési folyamat bizonyos szakaszait ciklikusan megismétlik. Két típusa létezik.

2.5.1. Inkrementális fejlesztés

Az evolúciós modellből alakul ki az 1980-as években, felhasználva a többi modell előnyeit is. Manapság is kedvelt fejlesztési modell. A modell a következő ábrával reprezentálható:



A rendszer nem teljes

Az inkremens egy kisméretű szoftverelem, amely valamelyest önálló egységet alkot a rendszeren belül. Egy-egy inkremens fejlesztése történhet például vízesésmodell segítségével is. Az inkremensek fejlesztése párhuzamosan haladhat, de az integráció és validálás csak egy inkremensre vonatkozhat adott időben. Az evolúciós modell előnyei hatványozottan jelentkeznek. Mindig egy működő rendszer van, már az első inkremens átadása után is. Természetesen csak az utolsó inkremens lezárása után válik teljessé a szoftver.

Az inkrementális fejlesztési folyamatnak számos előnye van:

1. A megrendelő már az első inkremens elkészülte után használatba veheti a szoftvert.
2. Az inkremensek, mint prototípusok információkat és tapasztalatokat adnak a későbbi rendszerinkremensek követelményeihez.
3. Kisebb a kockázata annak, hogy a teljes projekt kudarcba fullad. Az inkremensek többsége hiba esetén is nagy valószínűséggel leszállításra kerül a megrendelőnek.
4. Amennyiben a fontosabb rendszerkomponensek készülnek el előbb, akkor ezeket fogjuk a leggyakrabban tesztelni. Így jóval kisebb az esélye, hogy az ügyfélnél jelentkezik a hiba.

Mindezek ellenére itt is vannak problémák. Az inkremensek jó, ha nem nagyobbak 20 000 programsornál és minden egyes inkremensnek el kell látnia egy rendszerfunkciót. Tehát ajánlott határaink vannak, amelyeket össze kell hangolni a követelményekkel. Ezeket a határokat jó, ha betartja az ember. További gond adódhat, ha a szoftverben közös szolgáltatásokat használó inkremensek vannak, ugyanis csak az összes inkremens implementálása után lehetünk biztosak abban, hogy közös feladatokat ellátó rész rendben van.

2.5.2. Spirális fejlesztés

A spirális modellt Barry Boehm ajánlotta egy 1988-as cikkében. Nevét a fejlesztési folyamat reprezentálására szolgáló ábra alapján kapta. Az érdeklődők a következő internetes címen nézhetik meg az ábrát: http://en.wikipedia.org/wiki/Spiral_model.

Az ötlet az, hogy az életciklus minden egyes fázisára csináljunk egy ciklust és ne fázisokat ismételgessük. Ezek a ciklusok spirálszerűen kapcsolódnak egymáshoz. A spirál minden egyes körben a szoftverfolyamat egy-egy fázisát reprezentálja. Ez volt az első olyan modell, amely kifejezetten foglalkozott a kockázatkezeléssel. A kockázat lehet például egy új programozási nyelv használata, ahol egyrészt probléma lehet a tapasztalatok hiánya a

munkatársak részéről, másrészt pedig a nyelvhez kapcsolódó eszközök, például a fordító hatékonysága. A spirál minden egyes ciklusát négy részre oszthatjuk fel:

1. *Célok meghatározása.* Célok, megszorítások és követelmények meghatározása, azonosítása. A megfelelő kezelési terv kiválasztása a kockázati tényezők és az ezek miatt esetlegesen szükséges alternatív tervek figyelembevételével.
2. *Kockázatelemzés és –kezelés.* Minden felmerülő kockázati tényezőt részletesen elemezni kell és meg kell tenni a szükséges lépéseket a kockázat minimálisra csökkentéséhez. Problémás követelmények esetén prototípusrendszert is igénybe lehet venni a megoldás elősegítése érdekében.
3. *Fejlesztés és validálás.* Attól függően, hogy milyen feladattal állunk szemben, választani kell egy modellt és ennek segítségével végrehajtani a feladatot. Például felhasználói interfészekhez az evolúciós, biztonságtechnikai problémához formális és integrációhoz vízesés modellt lehet használni.
4. *Elemzés és tervezés.* Minden egyes kör végén eldöntjük, hogy tovább tudunk-e lépni. Ha igen, akkor elő kell készíteni a következő ciklust.

A modell egyre nagyobb hangsúlyt kap, különösen nagy rendszerek esetén jöhet jól, ahol a modell kockázatkezelésre fordított figyelme igen hasznos.

2.6. A választott fejlesztési modell

A fejlesztéshez a vízesésmodellt választom (de nem kizárólagosan), ugyanis elég tiszta és egyértelmű a követelményrendszer, továbbá egy kisméretű szoftverről van szó, amely nem haladja meg a 10 000 sort programkód tekintetében. A komponens alapú modell is megjelenik a projekt bizonyos részein, ugyanis korábbi projektjeimből is emelek át részeket.

3. Követelmények meghatározása

A rendszer követelményein a rendszer szolgáltatásainak és a rájuk vonatkozó megszorítások együttesét értjük. Nagyon fontos, hogy ebben a szakaszban nagyon alaposak és óvatosak is legyünk, mert ezen múlhat az egész projekt sikere. Sok projekt bukott már el nagyratörő terveken. Például a Windows Vista is ezért késett annyit annak idején.

3.1. Elképzelés (Vízió)

A szoftverfejlesztés akkor kezdődik, amikor valakinek a fejében megfogalmazódik egy gondolat, hogy egy adott tevékenységet vagy annak egy részét egy számítógépes programmal támogassa vagy hajtsa végre. Ez a gondolat természetesen csak nagy vonalakban tartalmazza azt, hogy a szoftvernek mit is kell tudnia. Az elgondolás sok esetben igen csak hiányos és egyes részei néha a valóságtól elrugaszkodottak. A szoftver tervezőjének a feladat, hogy ezekből az információkból egy használható követelményrendszer álljon össze, amelyre építve már kifejleszthető a szoftver. Egy valóságtól elrugaszkodott elképzelés felmerült a megrendelő részéről, amikor e dolgozat alapját képező projektet készítettem:

„Az oldal jó lenne, ha (automatikusan) fel tudna menni a hangszergyártók eredeti oldalára és onnan töltené le az egyes hangszerek, illetve hangtechnikai eszközök információit és ültesse át az oldal saját adatbázisába, hogy az oldal adminisztrátorainak minél kevesebb időt kelljen ezzel töltenie.”

Problémák ezzel az elvárással kapcsolatban:

1. Az oldalunk magyar nyelvű, az eredeti oldalak pedig általában angol nyelvűek. Nos, az automatikus fordítók - még a nagy cégek esetén is, akiknek jóval több pénzük van, mint adott esetben nekem vagy a megrendelőnek – messze nem tartanak ott, hogy ne legyen észrevehető a hibás fordítás.
2. Ha nincs is nyelvi akadály, még mindig igen nagy mesterséges intelligencia probléma, hogy hogyan találja meg a szükséges információt az oldalba épített ilyen irányú kereső. Egyrészt meg kell találnia a konkrét oldalt, ami tartalmazza az információt, másrészt a megtalált oldalon belül mit tekintsen releváns információnak. Ez emberi intelligenciát igényel, ahol - legjobb tudomásom szerint – nem tart a gépi intelligencia.

3. Amennyiben léteznének az előbbi két pontra megoldások, tartok tőle, hogy elég sokba kerülne egy ilyen jellegű projekt esetén, amit a megrendelő nemigen tudna és akarna megfizetni.

Ebből az egy óhajból is láthatjuk, hogy igen körültekintően kell a követelményeket meghatározni, ugyanis ha belemennék egy ilyen feltétel teljesítésébe, akkor azzal mind magunknak, mind az ügyfélnek kárt és csalódást okoznánk, ami a követelmények újragondolását igényelné és az eredeti ütemterv bukását vonná maga után.

3.1.1. A projekt esetén

Az elképzelés az volt, hogy egy dinamikus internetes oldalt kell létrehozni, amely a hangszerbolt megjelenését biztosítsa az interneten is. Az árusított termékeket meg lehessen tekinteni az oldalon is és különböző információk legyenek elérhetőek.

3.2. Megvalósíthatósági tanulmány

A szoftverfejlesztés és ebből kifolyólag a követelmények meghatározását is a megvalósíthatósági tanulmány készítésével ajánlott kezdeni. A tanulmány alapja (bemenete) az elképzeléskor megfogalmazott vázlatok, követelményelőzmények. Ahogy a nevében is benne van, a tanulmány célja az, hogy megmondja, rendelkezésre álló információk alapján, érdemes-e kifejleszteni a szoftvert, illetve kifejleszthető-e az adott keretek között.

Foglalkoznia kell azzal, hogy a kifejlesztendő rendszer támogatja-e az ügyfél üzletpolitikáját. Fel kell mérni a költségeket. Léteznek-e a szükséges technológiák a rendszerhez? A már létező rendszerhez hogyan illeszthető-e a szoftver és egyáltalán szükség van-e erre. A tanulmánynak meg kell mondania, hogy mit kell, és mit nem kell tudnia a leendő szoftvernek. Továbbá tartalmazhat javaslatokat, változtatási ajánlásokat. Amennyiben információra van még szükség, érdemes konzultálni a megrendelővel vagy annak kulcsfontosságú beosztottjaival. Két-három hét alatt el kell készülnie.

3.2.1. A projekt esetén

Mivel ez egy viszonylag kis projekt, ezért nem készült külön egy megvalósíthatósági tanulmány, mint dokumentum, ugyanis az elképzelések alapján át tudtam látni a rendszer lényeges elemeit. Ennek megfelelően világosan látszott számomra, hogy rendszer nem haladja

meg a képességeimet és az ügyfél lehetőségeit sem, tehát a rendszer kifejleszthető az adott kereteknek megfelelően.

3.3. Követelmények feltárása és elemzése

Amennyiben a megvalósíthatósági tanulmány pozitív választ adott, ez az a pont, ahol a követelményeket fel kell tárni és alaposan részletezni kell. E szakaszban a szoftvertervezők együtt dolgoznak a megrendelővel a kulcsfigurákon keresztül. A kulcsfigurák azok, akiknek közvetlenül közük lesz a rendszerhez, például a végfelhasználók, illetve bárki, akire hatással lesz az új szoftver.

A követelmények összegyűjtésének nehézségei:

- A kulcsfiguráknak legtöbbször fogalmuk sincs a számítástechnikáról és a mögöttes részről. Emiatt is van az, hogy vagy olyat kérnek, ami a valóságtól idegen, vagy nyitott kapukat döngetnek. Ilyen nyitott kapu például a Microsoft Office példája, ugyanis 80 %-ban olyan funkciókat kértek a felhasználók, amit a szoftver már tudott, csak nem találták meg. Többek között ezért is tervezték át a felhasználói felületet a 2007-es verzióban.
- A tervezőnek az adott szakterület sajátosságait el kell sajátítania, vagy legalábbis meg kell értenie. Ezek közé tartoznak a szakszavak, munkafolyamatok, stb.
- A kulcsfigurák különbözőképpen fejezik ki magukat, és ezek a nyilatkozatok átfedésben és ellentmondásban is állnak sokszor. Ezeket meg kell oldani.
- Politikai tényezők. Például egy vezető olyan funkciókat akar, amivel jobban tudja ellenőrizni, értékelni beosztottjai munkáját.
- A rendszernek egy adott gazdasági környezetben kell működnie, ami folyamatosan változik, különösen igaz ez napjainkban

A folyamat négy szakaszból áll: felderítés, osztályozás, prioritási sorrend felállítása és végül a dokumentálás. A követelmények összegyűjtésére, felderítésére többféle technika létezik:

- Nézőpont orientált megközelítés: a kifejlesztendő rendszer viszonylatában megnézzük, hogy milyen aspektusokból vizsgálható a rendszer.

    - Kulcsfigurák, akik belülről látják a rendszert, adatot állítanak elő és használnak fel. Az ellenmondások felszínre hozásához ez az egyik legmegfelelőbb módszer.
    - Rendszermodell oldali közelítés, ahol e modelleket felhasználva gyűjtjük össze a követelményeket.
    - Külső szemlélet: jellemzően a szolgáltatásokra és funkciókra fókuszál, például az interaktív rendszerek.
- Forgatókönyvek, amely az utóbbi időben egyre nagyobb hangsúlyt kapott. Mindig egy vagy néhány interakciót ír le (például: egy regisztráció menete egy weboldalon). Az elkészült forgatókönyvek egy interakció sorozatot alkotnak. A forgatókönyv a szoftver állapotainak egy sorozata. A sorozat tartalmazza az interakció normál menetét, a kivételes eseteket és annak kezelését, továbbá ha van, akkor a párhuzamosan lezajló eseményeket is. Az interakcióban lévő eseményeket kétféleképpen is feldolgozhatjuk:
    - Eseményorientált, azaz hogyan reagál a rendszer az eseményekre.
    - Használati esetek: külső szereplők és a rendszer viszonya.
- Az etnográfia a megfigyelésen alapul. A követelményfeltáró kimegy a helyszínre és figyeli, hogy hogyan zajlanak a munkafolyamatok. Eközben kiderül az is, hogy az egyes folyamatok hogyan hatnak egymásra.
- Interjúkészítés esetén a kulcsfigurákat kérdezzük ki személyesen.

Ezek után történik meg a követelmények osztályozása, sorba rendezése és egy olyan nem végleges követelmény dokumentum, amely arra alkalmas, hogy validáljuk és verifikáljuk a követelményeket. A validálás azt jelenti, hogy ellenőrizzük a megrendelővel közösen, hogy a követelmények a tényleges igényeit jelenítik meg. A verifikálás pedig azt, hogy ezeket a követelményeket teljesíteni is tudjuk.

3.3.1. A projekt esetén

A követelmény az volt, hogy egy internetes oldalt kell létrehozni, amely a következő funkciókat tartalmazza:

- Dinamikus weboldal legyen, azaz amit csak lehet az ügyfél vagy annak beosztottjai egy megfelelő felületen hozhassanak létre, illetve módosíthassanak.
- Az oldalnak hasonlítani kell egy korábban már létezett verzióhoz.

- A hangszerbolt által árusított termékeket meg lehessen tekinteni az oldalon úgy, hogy a termékről tartalmazzon bizonyos információkat. Például: név, ár, jellemzők, képek, stb.
- A termékek különböző csoportokba kerüljenek és ezek a csoportok további főcsoportokba, főkategóriákba tartozzanak.
- Az oldal látogatóinak legyen lehetőségük a saját (már használt) termékeiket eladásra felkínálni a többi látogatónak egyfajta hirdetés formájában, illetve legyen lehetőségük a funkciót használni arra is, hogy ne csak felkínáljanak, hanem keressenek is terméket. Azaz egy keres-kínál típusú funkció.
- Legyen egy üzenőfal is az oldalon.
- Legyen lehetőség hírlevél küldésére.
- Igazodjon a szabványokhoz az oldal.
- Azon a részeken, ahol a felhasználók adatokat küldhetnek az adatbázisba, anélkül hogy bármilyen azonosításon át kellene esniük, ott legyen egy egyszerű ellenőrző eszköz, amit gépek, internetes robotok nem tudnak átlépni, de az ember számára könnyű feladat.
- Amennyire csak lehet, ingyenes szoftvereken alapuljon az oldal.

3.4. Fogalomszótár

Az kölcsönös érthetőség miatt a legtöbbször szükség van egy fogalomszótárra is, ahol azon szakterület terminológiáját magyarázza meg a megrendelő, ahol szoftver működni fog.

A projektem esetén fogalomszótárra jelenleg nincs szükség, ugyanis nincsenek olyan szakterületi szavak, amelyeket ne ismerne egy átlagos ember. Ha későbbiekben bővül a szoftver, akkor előfordulhat, hogy szükség lesz rá. Annyit azonban érdemes megtenni, hogy a három különböző felhasználói csoportot megkülönböztessük:

1. *Látogató.* A legtöbb felhasználó ebbe a csoportba tartozik. Ők azok, akik böngészik az oldalt, az árusított terméket. Írhatnak az üzenőfalra, használhatják a Keres-Kínál funkciót, stb.
2. *Adminisztrátor.* A látogató szerepkörön felül hozzáférhet az adminisztrátori funkciókhoz az adminisztrátor oldalon keresztül. Viszont nem adhat ki közvetlenül

SQL parancsot és nem tekintheti meg a rendszerinformációs lapot az adminisztrátori oldalon. Tipikusan ők a megrendelők, azaz ebben az esetben a bolt tulajdonosai.

3. *Rendszergazda.* Mindenhez hozzáfér. A rendszer fejlesztője rendelkezik ezzel a joggal.

A követelmények hatással vannak a fejlesztő környezetre és ez a hatás visszafelé is igaz. Úgy gondolom, hogy ez egy alkalmas hely, hogy erről is szót ejtsek néhány oldalon.

4. Fejlesztői környezet, felhasznált technológiák

Talán meglepő lehet, hogy PHP és MySQL esetén miért nem a szokásos LAMP (Linux–Apache–MySQL–PHP) összeállítást választom, de Windows XP-vel dolgozom általában, ezért adta magát, hogy a beépített webszervert használjam, ugyanis ennek nincs semmilyen hatása arra, hogy a szoftver hogyan fog viselkedni a végleges helyén, az ügyfélnél. Ezek után lássuk a főbb összetevőket.

4.1. IIS

Az IIS az Internet Information Services rövidítése. Ez a Microsoft cég terméke. Feladata hálózati szolgáltatások megvalósítása. A szolgáltatások között található web kiszolgáló, FTP kiszolgáló, illetve levelezőszerver. Az IIS a Microsoft szerver rendszereinek alapja. A szoftver egy egyszerűbb, szolgáltatásokban korlátozott verziója a Windows bizonyos kiadásaiban (pl. Windows XP Pro) megtalálható a Windows összetevők között. A webhelyek közel 30 %-a használja. Első változata 1995-ben, a Windows NT 3.51-es változatához volt elérhető ingyenes kiegészítő formájában.

4.2. MySQL

A MySQL egy több felhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver. A szoftver fejlesztője a svéd MySQL AB cég, amely kettős licenceléssel teszi elérhetővé a MySQL-t; választható módon vagy a GPL, vagy egy kereskedelmi licenc érvényes a felhasználásra. 2008 januárjában a Sun Microsystems felvásárolta a céget. 2009-ben pedig az Oracle akarja a Sun céget felvásárolni. Ez az ügylet még nem zárult le, de ha a

hivatalok is rábólintanak, akkor a MySQL szempontjából érdekesen alakulnak a dolgok. A MySQL az egyik legelterjedtebb (több mint 6 millió telepítés) adatbázis-kezelő, aminek egyik oka lehet, hogy a teljesen nyílt forráskódú LAMP (Linux–Apache–MySQL–PHP) összeállítás részeként költséghatékony és egyszerűen beállítható megoldást ad dinamikus webhelyek szolgáltatására. 1995-ben készült az első verziója.

Egyedi illesztő felületekkel az adatbázis-kezelő elérhető több programozási nyelvből is. Egy MyODBC nevű ODBC interfész további, ODBC-t kezelő nyelvek számára is hozzáférhetővé teszi az adatbázis-kezelőt. A MySQL számára az ANSI C a natív nyelv. Továbbá számos operációs rendszeren érhető el, beleértve az elterjedteket is.

A MySQL 5.x képességei:

- ANSI SQL 99, számos kiegészítéssel
- Keresztplatformos elérhetőség
- Tárolt eljárások
- Adatbázis triggerek
- Kurzor adatbázisok
- "View" adatbázisok
- Valódi VARCHAR támogatás
- INFORMATION_SCHEMA támogatás
- "Strict" (szigorú) mód
- X/Open XA elosztott tranzakció-feldolgozás (DTP) támogatása; az Oracle InnoDB motorjának használata
- Különálló tároló motorok,(MyISAM olvasási sebességért, InnoDB a tranzakciókhoz és a referenciális integrációhoz, MySQL Archive az elavult adatok kevés helyen történő tárolására
- Tranzakciók az InnoDB, BDB és Cluster tároló motorokkal
- SSL támogatás
- Lekérdezés gyorstár (cache)
- Egymásba ágyazott SELECT -ek
- Szöveges indexelés és keresés a MyISAM motorral
- Beágyazott adatbázis-könyvtár

- Részleges Unicode támogatás
- ACID megfelelés az InnoDB-vel, BDB-vel és Cluster-rel
- Továbbfejlesztett MySQL Cluster
- "Példányosítás"

A következő képességekkel a MySQL rendelkezik, számos más relációs adatbázisrendszerrel ellentétben:

- Többféle tároló motor, amelyek között bármely táblához szabadon választhatunk
- Natív tároló motorok (MyISAM, Falcon, Merge, Memory (heap), MySQL Federated, MySQL Archive, CSV, Blackhole, MySQL Cluster, Berkeley DB, EXAMPLE, és Maria)
- Partnerek által fejlesztett tároló motorok (InnoDB, solidDB, NitroEDB, BrightHouse)
- Közösségi fejlesztésű tároló motorok (memcached, httpd, PBXT, Revision Engine)
- Akár egyéni tároló motor
- Commitek csoportosítása, több tranzakció fogadása többféle kapcsolatról, melyek meggyorsítják a tranzakciók lefolyását

4.3. HTML

A HTML (angolul: **H**yper**T**ext **M**arkup **L**anguage=hiperszöveges jelölőnyelv) egy leíró nyelv, melyet weboldalak készítéséhez fejlesztettek ki, és mára már internetes szabvánnyá vált a W3C (World Wide Web Consortium) támogatásával. Az aktuális változata a 4.01, mely az SGML általános jelölőnyelv egy konkrét alkalmazása (azaz minden 4.01-es HTML dokumentum egyben az SGML dokumentumszabványnak is meg kell, hogy feleljen). Ezt a tervek szerint lassan kiszorította volna az XHTML (ami a szintén SGML alapú XML leíró nyelven alapul), de mára nyilvánvaló vált, hogy a HTML 5 veszi át a helyét és az XHTML vonalat nem folytatják.

4.4. PHP

A PHP (PHP: Hypertext Preprocessor) nyílt forráskódú, számítógépes szkriptnyelv, legfőbb felhasználási területe a dinamikus weboldalak készítése. A nyelvet eredetileg Rasmus Lerdorf alkotta meg 1994-ben, de a ma létező egyetlen PHP implementációt már a PHP

Group tartja karban és fejleszti. A PHP a saját licensze alatt kerül kiadásra, a Free Software Foundation így szabad szoftverként tartja számon. A PHP a legtöbb webszerverre, operációs rendszerre és platformra ingyenesen telepíthető. Manapság több mint 20 millió weboldal és egymillió szerver futtat PHP-t.

A PHP fejlődése kezdetén csak CGI-programok halmaza volt. Ezeket Lerdorf néhány Perl szkript lecserélésére írta, amelyeket honlapjának karbantartására (például önéletrajzának megjelenítésére és a látogatottság mérésére) használt. Később ezeket a programokat kombinálta a szintén általa írt Form Interpreter (űrlap-értelmező) alkalmazással - így jött létre a PHP/FI (Personal Home Page / Forms Interpreter), ami már jóval szélesebb funkcionalitással bírt. Az új, C nyelven megírt változat képes volt adatbázisokhoz kapcsolódni és segítségével egyszerű dinamikus weboldalakat is létre lehetett hozni. Lerdorf 1995. június 8-án adta ki a PHP első nyilvános változatát (PHP 2).

1997-ben Zeev Suraski és Andi Gutmans, két izraeli fejlesztő újraírta az értelmezőt, ezzel megteremtve a PHP 3 alapját - ekkor született meg a PHP új neve, a PHP: Hypertext Preprocessor rekurzív rövidítés is.

A két fejlesztő ekkor alapította meg a Zend Technologies-t is, ami máig aktívan ellenőrzi a PHP fejlesztését. A Zend Engine 1.0 által hajtott PHP 4 2000. május 4-én jelent meg. Ezt követte 2004. július 13-án a következő nagy mérföldkőnek számító, az új Zend Engine II-n alapuló PHP 5. Az ötös verzió sok újítást tartalmazott: fejlettebb objektumorientált programozási lehetőségeket, a PDO (PHP Data Objects) adatbázis-absztrakciós kiterjesztést, és sok teljesítményt növelő javítást is.

A PHP oldalak elkészítésénél a HTML-t gyakorlatilag csak mint formázást használják, ugyanis e lapok teljes funkcionalitása a PHP-re épül. Amikor egy PHP-ben megírt oldalt akarunk elérni, a kiszolgáló először feldolgozza – egy interpreter segítségével - a PHP utasításokat, és csak a kész (HTML) kimenetet küldi el a böngészőnek, így a programkód nem is látható kliens oldalról.

A PHP nyelv lényegében nagymértékű kiegészítése a HTML-nek, ugyanis rengeteg olyan feladat végezhető el vele, amelyre az ügyféloldali szkriptek nem képesek (vagy ha igen, korlátozottan). Minden olyan esetben, ahol nagyszámú ismétlődő feladatsort kell végrehajtani

(például képek listázása és linkelése, listakészítés stb.), ott ez a programnyelv nagyszerű segítség.

4.5. JavaScript

A JavaScript programozási nyelv egy objektumalapú szkript nyelv, amelyet weblapokon elterjedten használnak. Eredetileg Brendan Eich, a Netscape Communications mérnöke fejlesztette ki; neve először Mocha, majd LiveScript volt, később „JavaScript" nevet kapott, és szintaxisa közelebb került a Sun Microsystems Java programozási nyelvéhez. A JavaScriptet először 1997–99 között szabványosította az ECMA „ECMAScript" néven. A jelenleg is érvényes szabvány az ECMA-262 Edition 3 (1999. december), ami a JavaScript 1.5-nek felel meg. Ez a szabvány egyben ISO szabvány is.

4.6. CSS

A CSS (angolul Cascading Style Sheets) a számítástechnikában egy stílusleíró nyelv, mely a HTML vagy XHTML típusú strukturált dokumentumok megjelenését írja le. Ezen kívül használható bármilyen XML alapú dokumentum stílusának leírására is, mint például az SVG, XUL stb. A CSS-t a weblapok szerkesztői és olvasói egyaránt használhatják, hogy átállítsák vele a lapok színét, betűtípusait, elrendezését, és más megjelenéshez kapcsolódó elemeit. A tervezése során a legfontosabb szempont az volt, hogy elkülönítsék a dokumentumok struktúráját (melyet HTML vagy egy hasonló leíró nyelvben lehet megadni) a dokumentum megjelenésétől (melyet CSS-sel lehet megadni).

5. Tervezés

Miután megvannak a követelmények, hozzáláthatunk a rendszer megtervezéséhez. Nagy rendszerek esetén érdemes szétbontani a rendszert több alrendszerre. Mivel a megvalósított alkalmazás nem egy nagy rendszer, így csak két alrendszert különböztetek meg, normál oldal és adminisztrátor oldal, amelyeket több kisebb modul épít fel. Nagyon fontos, hogy a tervezés jó legyen. Ha jól tervezzük meg rendszert, azt el lehet rontani, de egy rosszul megtervezett rendszert jól implementálni lehetetlen.

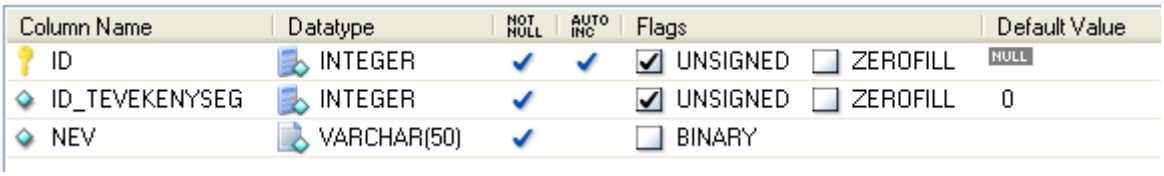
A tervezés sokféleképpen történhet. Használhatunk különböző modelleket, strukturált módszertanokat, UML-t vagy valamilyen informális jelölésrendszeren alapuló vázlatot. Itt is elmondható, hogy nincs senki, aki meg tudná mondani a pontos receptet. A tervezést létező tervek vizsgálatával és saját tervünk másokkal történő megvitatásával tanulhatjuk. Részemről az adatbázis megtervezésével szoktam kezdeni az ilyen jellegű projektek esetén, ugyanis az adatbázis meghatározása után már sokkal jobban átlátom a rendszert és könnyebb a továbbhaladás.

5.1. Adatbázis tervezése

Az adatbázis tervezése egy sarkalatos pontja az alkalmazás sikeres kifejlesztésének. Elmondható, hogy nagy tapasztalat kell hozzá, hiszen könnyű elrontani. Messze nem elég hozzá, ha valaki csak programozni tud, de nem tanult még az adatbázisokról elmélyülten, mert ez esetben garantált, hogy rosszul fog hozzá ehhez a részhez. Sajnos már én is találkoztam olyan esettel, amikor egy ehhez hasonló alkalmazást a fejlesztője egy adatbázistáblával kívánt megoldani. Nyilván, nem lehetetlen így megoldani, de tudni kell, hol a határ az „egyszerűség” és szakszerűség között. Sajnos nem is nagyon akarta megérteni, hogy miért nem lesz az jó úgy. Nyilván nem bántásként teszek erről említést, inkább intő példaként, hiszen a szakmai alázat itt is szükséges, ha eredményesen akarunk dolgozni.

A weboldal számára összesen hét adatbázis táblára lesz szükségünk.

CSOPORT tábla, amely a különböző típusú csoportok neveit tartalmazza a négy fő kategórián belül. Például a Hangszerek fő kategórián belül a Gitárok, Orgonák, stb. csoportok.



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL
ID_TEVEKENYSEG	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	0
NEV	VARCHAR(50)	✓		☐ BINARY	

Mezőlista:

- ID: A csoport egyedi azonosítója és elsődleges kulcsa a táblának. A KESZLET tábla hivatkozik erre a táblára a mező alapján.
- ID_TEVEKENYSEG: Meghatározza, hogy a csoport melyik fő kategóriához tartozik. Mivel a fő kategóriák fixek, így ezek számára nincs külön adatbázistábla.
- NEV: A csoport neve (pl. Gitárok)

GYARTO tábla, amely gyártókról tartalmaz információkat. Igaz, hogy egyelőre csak a gyártó nevét tartalmazza, de ha a későbbiekben szükséges, akkor könnyedén bővíthető például egy rövid leírással a gyártóról, vagy egy internet cím mezővel.

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL
NEV	VARCHAR(50)	✓		☐ BINARY	

Mezőlista:

- ID: A gyártó egyedi azonosítója és elsődleges kulcsa a táblának. A KESZLET tábla hivatkozik erre a táblára a mező alapján.
- NEV: A gyártó neve (pl. Yamaha)

KESZLET tábla, amely tartalmazza mindazon árucikkeket, melyeket a bolt árul.

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL
ID_CSOPORT	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	0
ID_GYARTO	INTEGER	✓		☑ UNSIGNED ☐ ZEROFILL	0
NEV	VARCHAR(100)	✓		☐ BINARY	
JELLEMZOK	TEXT				NULL
KEPNEV1	VARCHAR(50)			☐ BINARY	
KEPNEV2	VARCHAR(50)			☐ BINARY	
KEPNEV3	VARCHAR(50)			☐ BINARY	
KEPNEV4	VARCHAR(50)			☐ BINARY	
KEPNEV5	VARCHAR(50)			☐ BINARY	
KEPNEV6	VARCHAR(50)			☐ BINARY	
ERTEK	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL
LOGIKAI	SMALLINT(5)	✓		☑ UNSIGNED ☐ ZEROFILL	0

Mezőlista:

- ID: A készletelem, árucikk egyedi azonosítója és elsődleges kulcsa a táblának.
- ID_CSOPORT: A CSOPORT táblára hivatkozik, azaz megmondja, hogy a készletelem melyik csoportba tartozik egy főkategórián belül.
- ID_GYARTO: A GYARTO táblára hivatkozik, azaz megmondja, hogy a terméket melyik cég gyártotta.
- NEV: A termék neve.
- JELLEMZOK: A termék részletes leírása.
- KEPNEV1, …, KEPNEV6: Minden termékhez hat képet lehet hozzárendelni. A képek a „kepek” alkönyvtárba kerülnek elhelyezésre feltöltéskor. A feltöltést az adminisztrátori oldalon lehet elvégezni. A képek neve a következő képlet alapján adódik: KEPNEVi = keszlet_{KESZLET.ID}_{i – 1}. A kapcsos zárójelek közti részekből adódó értéket karakterláncként kell figyelembe venni. A kiterjesztés tetszőleges, tehát az adminisztrátor felelőssége, hogy mit tölt fel képként. Példa (jpg kiterjesztésű fájl esetén): i = 3, KESZLET.ID = 12. Ekkor KEPNEV3 = keszlet_12_2.jpg
- ERTEK: Az áru bruttó értéke.
- LOGIKAI: Bitenként tárol logikai értékeket.
    1. 0: Alapértelmezés. Semmilyen extra információ nincs.

2. 1. bit be van kapcsolva: A termék akciós.
3. 2. bit be van kapcsolva: A termék jelenleg nem kapható, tehát a felhasználói oldalon nem látható.
4. …

KERESKINAL tábla tartalmazza a felhasználók által keresett vagy kínált termékeket. Tehát az oldal lehetőséget biztosít arra, hogy a felhasználók is bizonyos szintű kereskedést folytassanak az oldal segítségével.

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL
NEV	VARCHAR(60)			☐ BINARY	NULL
EMAIL	VARCHAR(200)			☐ BINARY	NULL
TERMEKNEV	VARCHAR(100)			☐ BINARY	NULL
AR	DOUBLE			☐ UNSIGNED ☐ ZEROFILL	NULL
LOGIKAI	SMALLINT(5)	✓		☑ UNSIGNED ☐ ZEROFILL	0
KEPNEV1	VARCHAR(50)			☐ BINARY	
KEPNEV2	VARCHAR(50)			☐ BINARY	
KEPNEV3	VARCHAR(50)			☐ BINARY	
KEPNEV4	VARCHAR(50)			☐ BINARY	
KEPNEV5	VARCHAR(50)			☐ BINARY	
KEPNEV6	VARCHAR(50)			☐ BINARY	
MEGJEGYZES	TEXT				NULL
DATUMIDO	DATETIME				NULL

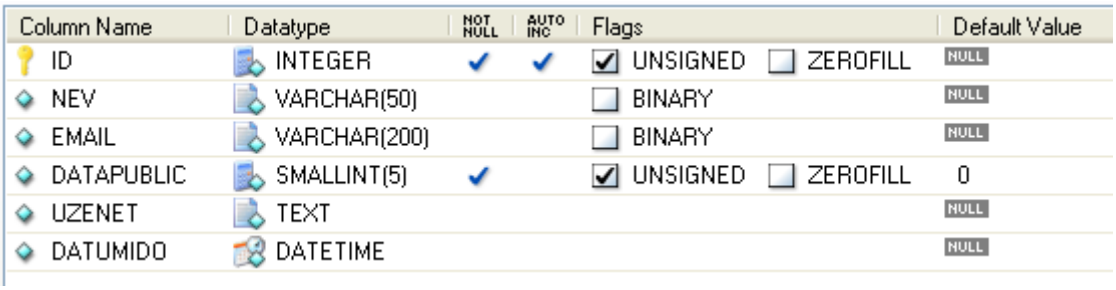
Mezőlista:

- ID: A felkínált vagy keresett árucikk egyedi azonosítója és elsődleges kulcsa a táblának.
- NEV: A kereső vagy kínáló felhasználó neve.
- EMAIL: A kereső vagy kínáló felhasználó e-mail címe.
- TERMEKNEV: A keresett vagy kínált termék neve.
- AR: A termék ára.
- LOGIKAI: 0: A terméket keresi a felhasználó, 1: A terméket kínálja a felhasználó.
- KEPNEV1, …, KEPNEV6: Minden termékhez hat képet lehet hozzárendelni. A képek a „kepek” alkönyvtárba kerülnek elhelyezésre feltöltéskor. A képek neve a következő képlet alapján adódik: KEPNEVi = kereskinal_{KERESKINAL.ID}_{i – 1}. A kapcsos zárójelek közti részekből adódó értéket karakterláncként kell figyelembe

venni. A kiterjesztés tetszőleges kivéve a php kiterjesztés. Példa (jpg kiterjesztésű fájl esetén): i = 3, KERESKINAL.ID = 12. Ekkor KEPNEV3 = kereskinal_12_2.jpg

- MEGJEGYZES: A termék leírása, megjegyzések.
- DATUMIDO: A hirdetés feladásának ideje.

UZENOFAL tábla. Az oldal üzenőfalának hozzászólásait tárolja.



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
ID	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL
NEV	VARCHAR(50)			☐ BINARY	NULL
EMAIL	VARCHAR(200)			☐ BINARY	NULL
DATAPUBLIC	SMALLINT(5)	✓		☑ UNSIGNED ☐ ZEROFILL	0
UZENET	TEXT				NULL
DATUMIDO	DATETIME				NULL

Mezőlista:

- ID: Az üzenetek egyedi azonosítója és elsődleges kulcsa a táblának.
- NEV: A üzenetet küldő felhasználó neve.
- EMAIL: A üzenetet küldő felhasználó e-mail címe.
- DATAPUBLIC: Logikai értékeket tárol bitenként.
    1. 0: alapértelmezés, azaz semmilyen adata nem nyilvános és az üzenet normál állapotban van.
    2. 1. bit be van kapcsolva: Az e-mail cím nem nyilvános.
    3. 2. bit be van kapcsolva: Az üzenet moderálva lett.
    4. 3. bit be van kapcsolva: Az üzenet logikailag törölt, de az adminisztrátorok még olvashatják.
    5. 4. bit be van kapcsolva: Az üzenet fizikailag törölt, azaz ténylegesen, visszaállíthatatlanul törölve lett, senki sem tud hozzáférni.
- UZENET: Az üzenet szövege.
- DATUMIDO: Az üzenet elküldésének időpontja.

HIRLEVEL tábla. Az oldal hírlevelére feliratkozottak e-mail címét tárolja.

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
EMAIL	VARCHAR(200)	✓		BINARY	

A tábla jelenleg egy mezőt tartalmaz, mely egyben az elsődleges kulcs is. A későbbiekben, ha szükséges, bővíthető további mezőkkel.

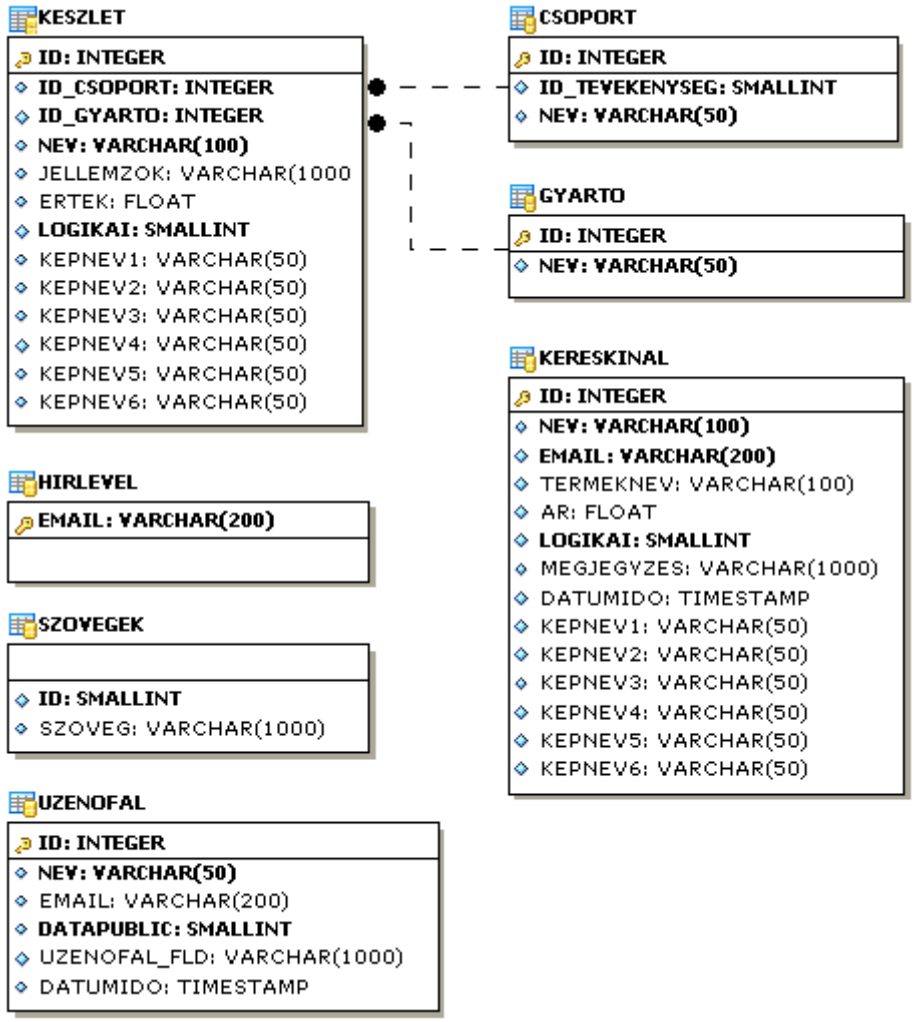
SZOVEGEK tábla. A megrendelő, a rugalmasságra való tekintettel, azt kívánta, hogy az oldal bizonyos részeit (Cégismertető, Szolgáltatások, stb.) szintén szerkeszthesse és ne egy előre megszerkesztett HTML oldal legyen.

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
ID	SMALLINT(5)	✓		☑ UNSIGNED ☐ ZEROFILL	0
SZOVEG	TEXT				NULL

Mezőlista:

- ID: Az elsődleges kulcsa a táblának. Továbbá megadja, hogy mely funkcióhoz, oldalhoz tartozik a rekord. Ezek az azonosítók előre rögzítve vannak és a következők:
    1. Cégismertető oldal
    2. Szolgáltatások oldal
    3. Hitel oldal
    4. Partnerek oldal
    5. Újdonságok oldal
- SZOVEG: A funkcióhoz tartozó oldal szövege, tartalma, amely HTML elemeket is tartalmazhat.

Kapcsolat a táblák között:



Két kapcsolat létezik a táblák között, amelyek 1:N típusú kapcsolatok.

1. A KESZLET és a CSOPORT tábla között áll fent.
2. A KESZLET és a GYARTO tábla között áll fent.

A kapcsolatok leírását már kifejtettem a három tábla megfelelő mezőinél.

5.2. Az alkalmazás alrendszerei, moduljai

Miután elkészült az adatbázisterv hozzákezdhetek az alkalmazás alrendszerre és azokon belül modulokra bontására. Figyelembe véve a követelményekben meghatározott részleteket kialakul az oldal felépítése és a modulok közötti kapcsolat- és vezérlésrendszer. Első lépésben a két alrendszer váza kerül meghatározásra. A normál oldal, ami a látogatóknak szól és az adminisztrátor oldal, ahol az oldal adatait lehet menedzselni. Természetesen azt is lehet, hogy nincs így elkülönítve egymástól, ízlés és alkalmazás kérdése, hogyan oldja meg a fejlesztő, illetve melyiket célszerű használni. De talán biztonságosabb, ha külön vannak választva. Ezek után következik a modulok megtervezése.

Az alrendszerek és modulok nem különböztethetők meg egyértelműen, de érdemes azokat a következő módon elképzelni:

1. Egy alrendszer egy olyan önálló rendszer, amely működése nem függ más alrendszerek szolgáltatásaitól. Az alrendszerek modulokból épülnek fel, és az egyéb alrendszerekkel interfészeken keresztül kommunikálnak. Jelen esetben ez az interfész az adatbázis.
2. Egy modul olyan rendszerkomponens, amely más modulok számára szolgáltatás(oka)t biztosít. Általában nem tekintjük önálló rendszernek.

Az alrendszerek modulokra bontása során két fő stratégia ismeretes:

1. Objektumorientált felbontással a rendszert egymással kommunikáló objektumok halmazára bontjuk fel.
2. Funkcióorientált csővezetékek használatával a rendszert bemenő adatok elfogadó és azokat kimenő adatokká alakító funkcionális modulokra bontjuk. Az általam használt modell ehhez hasonlít.

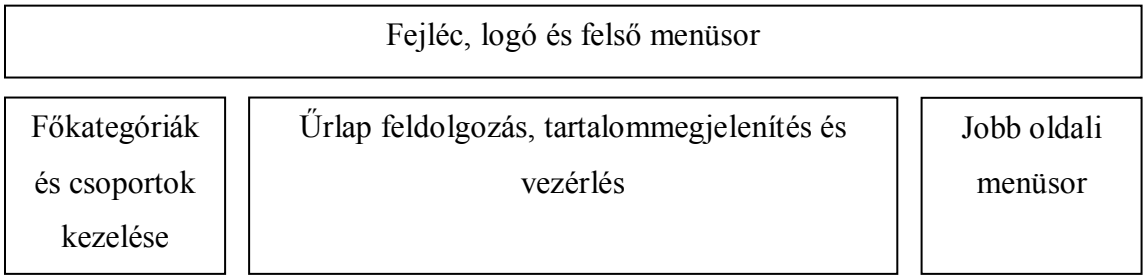
Az objektumorientált megközelítésben a modulok egyéni állapottal rendelkeznek, valamint az ezeken az állapotokon értelmezett műveletekkel rendelkező objektumoknak felelnek meg. A csővezetékelvű modellben a modulok funkcionális transzformációk. A modulok mindkét esetben szekvenciális komponensként vagy folyamatként valósíthatók meg.

A modulok tervezése abból áll. hogy meghatározzuk, mely funkciók megvalósításáért felelnek, továbbá hogyan kapcsolódnak a rendszerhez és más modulokhoz. Egy ilyen,

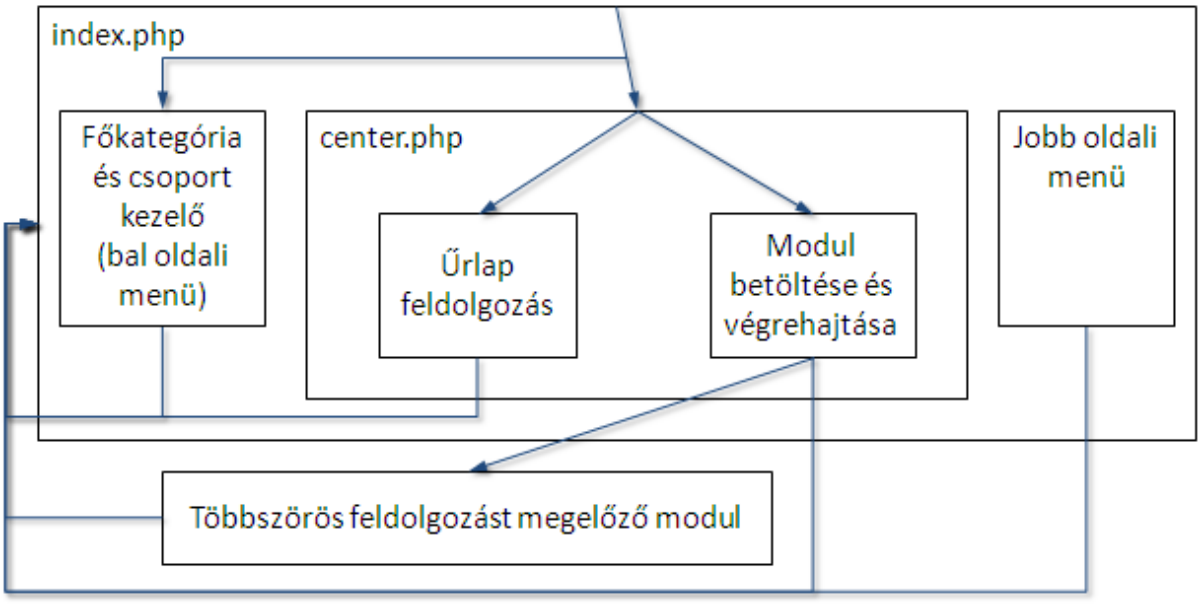
viszonylag kis rendszer esetén az adatbázistervből, a követelményekből és tapasztalatainkat felhasználva ez nem túl bonyolult munka. A következőkben lássunk néhány szemléltető ábrát az alkalmazás megértéséhez.

5.2.1. Normál oldal

Felületterv:



Adat és vezérlési folyamatok modellje:

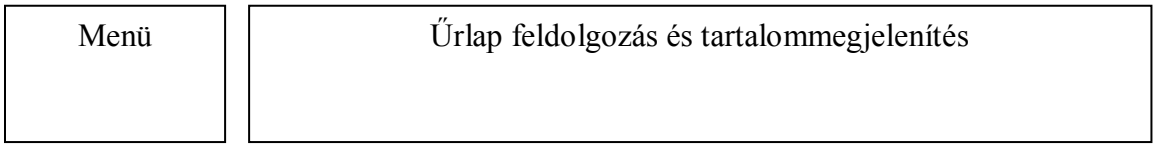


A többszörös feldolgozást megelőző modul olyan esetekben szükséges, amikor fennáll a veszélye annak, hogy egy felhasználó által feldolgozásra elküldött űrlap valaminek következtében többször is elküldésre kerül a felhasználó akarata ellenére. A legtöbb böngésző figyelmeztet arra, hogy újra akarjuk küldeni az űrlapot, de van olyan, amelyik nem. Ez a kellemetlenség tipikusan akkor fordulhat elő, ha ugyanaz a neve feldolgozó oldalnak és az

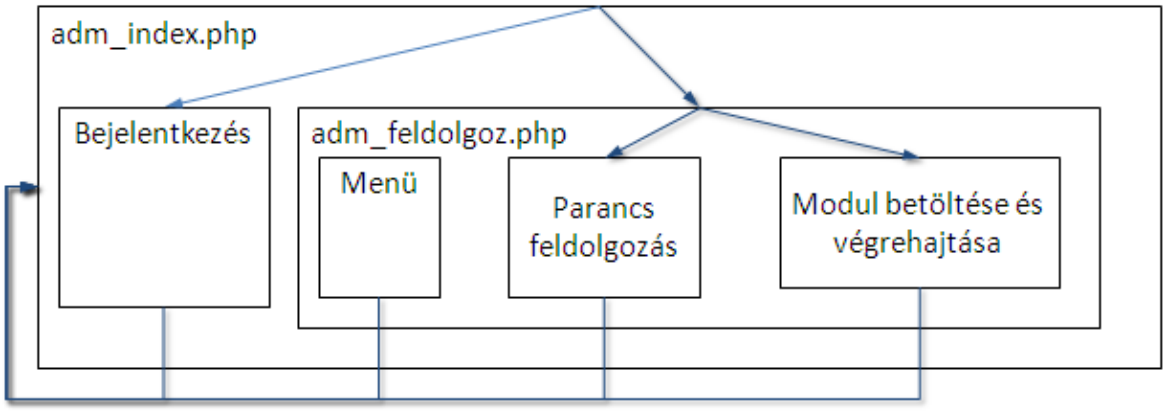
űrlapot tartalmazó oldalnak, függetlenül attól, hogy forráskód szintjén különböző fájlokban vannak, mert az include PHP funkció által beilleszthetünk más forrásfájlokat is a kódunkba.

5.2.2. Adminisztrátor oldal

Felületterv:



Adat és vezérlési folyamatok modellje:



5.3. Biztonság

Az internetről elérhető (egy weboldal egyértelműen ilyen) szoftverrendszerek folyamatos veszélynek vannak kitéve a rosszindulatú támadók miatt. A támadók kárt tehetnek a rendszer hardverében, bizalmas adatokat szerezhetnek meg, vagy akadályozhatják a rendszer által nyújtott szolgáltatásokat. A szoftvert fel kell készíteni arra (amennyire rajta áll), hogy ezeket a támadásokat, veszélyeket kivédje.

Nincsenek kötött, kőbe vésett szabályok arra vonatkozóan, hogyan lehet elérni egy adott rendszer biztonságát. Az egyes rendszereket különböző technikai eljárások és felhasználói csoportok jellemzik (pl. egy banki rendszernél jóval nagyobb biztonságot várunk el, mint egy fórumon), de vannak olyan általános irányelvek, amelyeket széles körben alkalmazhatunk. Tipikus problémák lehetnek a következők:

5.3.1. Bemenet ellenőrzése

Gyakori támadástípusnak számít, amikor a támadók váratlan bemenetet adnak a rendszernek (például szám helyett karakterláncot), aminek következtében az kiszámíthatatlan viselkedést produkálhat. Ez vezethet rendszerösszeomláshoz, amikor a szolgáltatások elérhetetlenné válnak, de az is elképzelhető, hogy a bemenet ártalmas kódot tartalmaz, amit a rendszer lefordít. A hosszú bemeneti sztringek puffertúlcsorduláshoz vezethetnek. Ennek kihasználása már 20 éves múltra tekint vissza. Az ún. SQL-beoltás, amely során nem kívánt SQL utasítást próbálnak belecsempészni a bemenetbe és ezáltal akarnak az üzemeltetők számára nem kívánt célt elérni (pl. jogokhoz jutni).

Ezen problémák nagy részét megelőzhetjük, ha validáljuk a rendszer összes lehetséges bemenetét. Nagyon fontos, hogy egyetlen bemenet se maradjon valamilyen ellenőrzés nélkül. A legtöbb bemenetre megadható egy ésszerű hosszkorlát, amelyet nem léphet át semmiképpen sem. Például senkinek sem hosszabb a neve 100 karakternél.

Abban az esetben, ha nem annyira fontos, hogy explicit módon foglalkozzunk a hibakezeléssel, akkor az is megoldás lehet, hogy típuskényszerítést alkalmazunk minden esetben. Ez tipikusan akkor lehet hasznos, amikor számot várunk bemenetként. Ha a bemenettel nincs probléma, akkor a típuskényszerítés sértetlenül hagyja a bemenetet, ha viszont ártó szándékú a bemenet, akkor legfeljebb nem azt a számot kapjuk, amit kellene, de ez nem okoz olyan problémát, amit ne lehetne elfogadhatónak tartani. Például egy fórum lapozásánál a rendes felhasználó nem is fut bele a problémába, a támadó meg megérdemli, hogy küldje el újra a lapozási kérelmét és ne trükközzön azzal, hogy megpróbálja feltörni az oldalt. Tehát fontos szempont, hogy az ilyen jellegű hibakezelés a rendes felhasználóknak ne okozzon felesleges terheket.

Az SQL-beoltás ellen úgy védekezhetünk a leghatékonyabban, ha az SQL utasítást mindig megfelelően előkészítve küldjük el az adatbázis kiszolgálónak.

Példa SQL-beoltásra (Jelszó átírása):

```
<?php
$lekerdezes =
     "UPDATE users SET jelszo='$pwd' WHERE uid='$uid';";
?>
```

A rosszindulatú felhasználó a *' or uid like'%admin%'; --* értéket adja át a $uid változónak, és ezzel "megváltoztatja az adminisztrátor jelszavát. Ezt az SQL parancsot ezek így ferdítik el:

```
<?php
$query = "UPDATE users SET pwd='...' WHERE uid='' or uid like
     '%admin%'; --';";
?>
```

5.3.2. Ember vagy gép

Itt nem is arról van szó, hogy csak emberek használják a weboldalt és gépek, internetes robotok, programok nem használhatnák. Vannak olyan programok, amelyek nemcsak indexelik a weboldalakat a későbbi keresés érdekében, hanem felesleges információkkal vagy nem kívánt reklámokkal szemetelik tele az oldal vendégkönyvét, regisztrációs oldalát vagy olyan részeit, amely nem kíván magasabb szintű hozzáférést.

Általában a „**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part" (teljesen automatizált nyilvános Turing-teszt a számítógép és az ember megkülönböztetésére) módszer valamilyen változatát használják védelem gyanánt. A tesztmódszer rövidítve: CAPTCHA. A teszt során a számítógép generál egy feladványt, amit csak egy ember tud helyesen megválaszolni, de a válasz helyességét a gép is könnyedén el tudja dönteni.

A projektemben az oldalak védelmére a következő egyszerű, de eddigi tapasztalataim alapján igen hatékony algoritmust használom:

1. Az alkalmazás véletlenszerűen választ egy maximum háromjegyű pozitív egész számot. Az alkalmazásban van egy fix karaktersorozat, amelyhez hozzáfűzzük ezt a számot. Így kapunk egy ellenőrző szrtinget. Előállítja a sztring hash (pl. MD5) kódját. A sztring hash kódját és a szám magyar nyelvű szövegét beleírja a felhasználónak elküldendő oldal megfelelő űrlapjába.
2. A felhasználó megkapja az oldalt a böngészőben és látja a számot szövegesen leírva, a látogatónak a megfelelő helyre be kell írnia a szám számjegyes alakját. Az oldalban benne van a hash kódja az ellenőrző sztringnek, amely a többi küldött adattal visszakerül az alkalmazáshoz. Ha a felhasználó ember, akkor könnyedén be tudja írni

a számot, ha viszont egy robot, akkor eddigi tapasztalataim szerint nem fogja tudni megoldani a feladványt.

3. A felhasználó elküldi az űrlapot feldolgozásra az alkalmazásnak. Az alkalmazás az előbb említett fix karaktersorozathoz hozzáfűzi, a felhasználó által küldött számot. Az alkalmazás ellenőrzi, hogy az így kapott karaktersorozat hash kódja megegyezik-e a kapott hash kóddal. Csak akkor lesz feldolgozva az űrlap, ha a két hash kód egyezik. Ha nem egyeztek a kódok, akkor hibaüzenettel visszatérünk arra az oldalra, ahonnan az űrlap jött.

Látható, hogy a biztonság azon a bizonyos fix karaktersorozaton múlik. A fix karaktersorozat az alkalmazás kódjában, jelen esetben a PHP kódban van, így csak az férhet hozzá, aki a kódhoz is, tehát biztonságos. Eddigi tapasztalataim alapján egyszer sem törték még át ezt a védelmet. Úgy gondolom, hogy ez a siker a magyar nyelv kevésbé elterjedtségének is betudható, ugyanis a legtöbb ilyen ártó szándékú robotot nem magyarok készítik és ilyen védelemre nincsenek felkészülve ezek a programok. Nyilvánvaló, ha ez megváltozik, akkor valamilyen bonyolultabb módszer szükséges, illetve ezt az eljárást kell tovább fejleszteni.

5.4. Felhasználói felületek

Általában a legtöbb ügyfélnek arról már határozott elképzelése van, hogy hogyan nézzen ki az alkalmazás. Továbbá a felhasználói felület tervezése, a színek meghatározása, logó tervezése nem kifejezetten az informatikus feladata. Erre ott vannak a látványtervezők. A legtöbb esetben igaz az, hogy ritka az olyan informatikus, aki jó látványtervező is. A szoftvertervező, programozó feladata inkább az, hogy ezt a látványtervet segítsen az alkalmazásba integrálni. Esetleg néhány tanácsot ad, ha a helyzet úgy kívánja.

A működésének milyensége már kifejezetten az informatikus feladata, mert központi szerepe van, ugyanis ezen keresztül találkoznak a felhasználók a rendszerrel. Alapelv, hogy ne bosszantsuk fel a felhasználót a következetlen működéssel, azaz hasonló funkciók ugyanúgy működjenek. A felhasználót a lehető legjobban segíteni kell és nem akadályozni. Ebben nagy segítséget nyújt, ha megfelelő a súgórendszer, illetve egyértelmű a felület működése. Egy feladatot lehetőleg többféleképpen is meg lehessen oldani. Nyilvánvalóan sok

ötlet van, amelyeket összegyűjthetünk a saját tapasztalataink alapján is, amikor mi vagyunk a felhasználók.

5.5. Újrafelhasználás

A rendszerben több komponens is részben vagy egészben újrafelhasználáson alapul. Nem ez az első ilyen jellegű projektem, így amit hasznosnak találtam átemeltem a korábbi projektekből ide. Újrafelhasználáson alapszanak a következő komponensek:

- Az adminisztrátor oldal vezérlési szerkezete
- Az adminisztrátor oldal (My)SQL parancs végrehajtása funkciója.
- A Linkek funkció mind az adminisztrátori oldalon, mind a normál oldalon.
- Az Ember vagy gép fejezetben említett védelmi algoritmus.
- Az üzenőfal nagy része.
- Biztonsági eljárások

Az újrafelhasználás előnyei:

- Megbízhatóság növekedése, hiszen már több projektben bizonyított az újrafelhasznált elem.
- Csökkenthető kockázat, mert a rendszer újrafelhasználáson alapuló részeinek költségvonzatait már ismerjük.
- A szakterületi ismeretek be vannak építve a szoftverelembe.
- Általában követik a kialakult szabványokat.
- Fejlesztési idő radikálisan csökkenhet.

Hátrányok:

- Fel kell kutatni, katalogizálni és esetleg publikálni kell ezeket.
- Bízni kell abban, amit az újrafelhasználandó komponens állít magáról.
- Hiányos lehet a dokumentáció.

Nyilvánvalóan ezek a hátrányok saját komponenseknél nem jelentkeznek általában.

6. Implementáció és egységteszt

A szoftverfejlesztés implementációs szakasza nem más, mint a rendszer-specifikáció futtatható rendszerré történő konvertálása. Ez mindig magában foglalja a szoftver további tervezését és a programozást. A tervezés alatt azt kell érteni, hogy implementáció közben derülnek ki apróbb hibák és ezek visszahatnak az eredeti tervre, amelyet tovább kell finomítani ennek következtében.

Az implementáció nyilván nem csak azt jelenti, hogy elkészítjük az alkalmazás forráskódjait tartalmazó fájlokat, hanem mindent, ami a szoftverrendszerhez hozzátartozik: súgó állományok, telepítőprogram, dokumentáció, stb. Alkalmazása válogatja, hogy ezeknek mennyire részleteseknek kell lenniük.

Az egységteszt azt jelenti, hogy amikor elkészült egy modul vagy funkció, akkor alaposan leteszteli a fejlesztő, hogy jól működik-e, illetve megfelel-e a követelményekben meghatározott célkitűzéseknek. Ahány egység, annyi egységteszt van. Ez már a verifikációs folyamat része. Ezek a tesztek alfa tesztek is, ugyanis a rendszer még nem teljes funkcionalitású és ezekbe a tesztekbe általában még nem vonják be a megrendelőt.

Minden tesztre elmondható, hogy arra sajnos egyik sem alkalmas, hogy megmondja tökéletes-e szoftver, illetve a modul.

7. Integráció és rendszerteszt

A rendszer-integráció nem más, mint az egymástól függetlenül fejlesztett alrendszerek, modulok összeillesztése, hogy teljes rendszert alkossanak. Két lehetséges megközelítés van. Az egyik, hogy akkor történik meg az integráció, ha már minden alrendszer és modul elkészült. Az inkrementális integrálási módszer viszont technikai és szervezési oldalról is hasznosabb megközelítési mód, azaz a komponenseket itt egyenként integráljuk a rendszerbe.

Az inkrementális integrálási folyamata előnyei:

- A legtöbb esetben nem lehetséges az alrendszerek és modulok fejlesztését úgy ütemezni, hogy azok egyszerre készüljenek el. Ez az én esetemben nem merül fel, mivel egyedül fejlesztettem a projektet, de a valós életben ez ritka.
- Csökkenti a hibák lokalizációjának költségeit. Ha egyszerre integrálunk több alrendszert és modult, akkor a tesztelés folyamán fellépő hiba bármelyikben lehet. Amennyiben egyszerre csak egyet, akkor vagy abban lesz a hiba, vagy a most integrált modul és a rendszer egymásra hatása idézte elő.

Ha megtörtént az integráció vagy egy integrációs lépés, következik a rendszer tesztelése. A tesztelés célja a komponensek közötti interfészek és az egész rendszer működésének vizsgálata.

Azok az alrendszer-meghibásodások, amelyek általában a más alrendszerekre, modulokra tett érvénytelen feltételezéseken alapszanak, gyakran jönnek felszínre az integráció folyamatában. Mihelyt megjelenik egy ilyen probléma, az alrendszerekért felelős vezetőknek meg kell állapítaniuk, hogy ki a felelős a kialakult helyzetért. Ez sokszor egy elhúzódó folyamat. Ahogy egyre több rendszer integrál magába külső fejlesztők által készített hardver- és szoftverelemeket is, a rendszer-integráció egyre hangsúlyosabb szerepet kap.

A folyamat e részébe feltétlenül be kell vonni a megrendelőt is, hiszen addig nem kerülhet átadásra a szoftver, amíg az ügyfél el nem fogadja a rendszerteszt eredményét.

8. Működtetés és karbantartás

Miután lezajlott a teljes rendszerteszt és a megrendelő is elfogadja a szoftvert, megkezdődhet a rendszer tényleges, éles bevezetése az ügyfélnél. Ritka az, hogy a szoftverfejlesztő feleljen a működésért, azaz a megrendelőnél az erre kiválasztott szakembert betanítja arra, hogy milyen hardver és szoftver környezet szükséges a szoftverrendszer működéséhez. Megmutatja neki, hogy lehet konfigurálni, illetve biztosítani a rendszer folyamatos működését. Ha szükséges, a felhasználókat is betanítják a rendszer használatára. Amennyiben nem egy nagy rendszerről van szó, mint például ez esetben is, gyakran előfordul, hogy a felhasználó és a működtető ugyanaz a személy.

Mindig éles határ húzódik a szoftverfejlesztés folyamata és a szoftverkarbantartás között. A szoftverfejlesztés egy kreatív tevékenység, ahol a kezdeti elképzelésből egy munkarendszeren keresztül létrejön a szoftverrendszer. A szoftver karbantartása ezzel ellentétben a rendszeren történő változtatások folyamata, a rendszer üzembe állítása után. Annak ellenére, hogy a karbantartás költségei akár a fejlesztés költségeinek többszörösét is elérhetik, a karbantartási folyamat sokkal kisebb kihívásnak tekinthető, mint egy eredeti szoftver kifejlesztése. A telepítések utáni változtatások nem egyszerűen a szoftverhibák javítását jelentik. A változtatások nagy része az új követelmények következménye, amelyeket az üzleti és felhasználói igények változása váltott ki.

A karbantartásnak három típusát különböztetjük meg: a szoftverhibák kijavítására, a szoftver más működési környezethez történő adaptálására és a rendszer funkcionalitásának kibővítésére vagy módosítására irányuló karbantartás. A gyakorlatban ezek között nincs éles határvonal. Megfigyelések szerint a karbantartások 65 %-a az új követelmények implementálásával kapcsolatos, 18 %-a az új környezethez való adaptálással és 17 % a rendszer hibajavításával.

A fejlesztési és karbantartási költségek megoszlása alkalmazási területenként más és más. Például üzleti alkalmazások esetén kb. fele-fele az arány, beágyazott valós idejű alkalmazások esetén már a karbantartási költségek a négyszeresük is lehet a fejlesztési költségeknek.

9. Összefoglalás

Diplomamunkám célja az volt, hogy rálátást nyújtsak egy alkalmazás, azon belül is egy dinamikus internetes oldal létrehozásának folyamatára. Ahogyan már korábban is leírtam, a webes alkalmazások egyre nagyobb teret hódítanak az alkalmazások népes családjában. Ez a folyamat, úgy vélem, továbbra is folytatódni fog, hiszen a tendencia az, hogy a szoftverek mind jobban átköltöznek az internetre. A webes alkalmazásokat manapság ki sem nagyon kerülhetjük, ha az interneten böngészünk, hiszen nehéz olyan oldalt találni, amely csak statikus lapokból áll.

Természetesen a webes alkalmazások fejlesztésénél is nagyszerűen használhatók a hagyományos alkalmazásokhoz kifejlesztett módszertanok. E módszertanokról olvashattunk, a teljesség igénye nélkül, a második fejezetben. Láthattunk öt modellt azok előnyeit és hátrányait is kiemelve. Észrevettük, hogy nincs „csodaszer”, inkább csak ajánlások, hogy mikor melyiket célszerű választani, de a legtöbbször az igaz, hogy ezeket érdemes vegyíteni a hatékony munka érdekében. Taglaltam, hogy miért a vízesésmodell célszerű választás egy ilyen jellegű alkalmazáshoz.

Miután megvolt a hasznosnak talált fejlesztési modell, hozzákezdhettünk a modell alapján történő alkalmazásfejlesztéshez. Ennek alapján a harmadik fejezetben a követelmények meghatározásával, mint az első fontos résszel foglalkoztam. E szakasztól kezdődően igyekeztem az elmélet felől mind inkább a gyakorlat oldala felé is elmozdulni, így ettől fogva a fejlesztett alkalmazásra kivetíteni az elméleti fogalmakat. Itt dőlt el, hogy kifejleszthető-e a szoftver, ha igen, akkor jöhetett a ténylegesen követelmények meghatározása.

A követelmények meghatározásakor már nyilvánvalóan felmerül, hogy milyen környezetben fog működni, illetve fejlesztve a szoftver. A sokféle fejlesztőeszköznek köszönhetően lehet válogatni. Bemutattam egy környezetet, illetve összeállítást ezekből, amelyeket én is gyakran használok. Nyilván ez egy kissé kilóg a fejlesztés modellbeli menetéből, de fontos ez is, hiszen a fejlesztőkörnyezet, a felhasznált technológiák és a szoftver kölcsönösen hatnak egymásra, ezért jó volt erre is kitérni.

Ezek után hozzá lehettet kezdeni a tervezéshez. Láthattuk, hogy ez is egy olyan területe, a követelmény meghatározással együtt, a fejlesztésnek, amit ha rosszul csinálunk, garantáltan a sikertelen projektek népes halmazát gyarapítjuk. A tervezés is igen sokrétű feladat, de igyekeztem azokra a területekre tenni a hangsúlyt, amelyek egy ilyen jellegű internetes alkalmazásnál a legfontosabbnak tartok. Ezek voltak az adatbázis megtervezése, a vezérlési folyamatok és nem utolsó sorban a biztonság, amely hatványozottan fontos az internet világában.

A modellbeli következő három szakaszt nem tartottam szükségesnek olyan bőven kifejteni, hiszen az implementációt legjobban az alkalmazás forráskódjának vizsgálatával érthetjük meg. Az integrációt szintén a több tíz forráskód fájl áttekintésével, illetve azok összekapcsolásában láthatjuk a legjobban. A tesztélés tekintetében is csak a szükséges dolgokra szorítkoztam, ugyanis, ahogy már többször is említettem, ez nem egy nagy alkalmazás, emiatt könnyű tesztelni és sok tesztmódszerre ezek miatt nincs is szükség. Az utolsó részben röviden láthattuk, hogy a szoftver élete igazán csak akkor kezdődik meg, amikor a felhasználók birtokba veszik és elkezdik használni. Láthattuk azt is, hogy sok szoftver esetében a fejlesztés ezzel nem zárul le, hanem karbantartás formájában folytatódhat, ha felek ezt úgy kívánják. Kitértem arra is, hogy a karbantartás nem feltétlenül kisebb horderejű folyamat, mint az addigi fejlesztés, ugyanakkor sok szempontból egyszerűbb.

Eddigi tapasztalataim sokat segítettek abban, hogy az alkalmazás viszonylag könnyen és gyorsan elkészüljön, viszont a diplomamunkához az is kell, hogy meg is írjuk hozzá ezt a dolgozatot, ami kicsit nehezebben ment, de úgy gondolom sikerült ezt az akadályt is teljesíteni. Természetesen ez a dolgozat csak egy ízelítőt tud adni arról, hogy is folyik egy webes alkalmazás fejlesztése, ezért igyekeztem néhány példával illusztrálni, hol kell nagyon odafigyelni, ha ilyen projektekbe fogunk. Remélem, ez sikerült.

Mivel a webes alkalmazások egy fejlődő ága az informatikának, így a további pályafutásom alatt is foglalkozni kívánok ezzel a területtel, más technológiák megismerése által is.

10. Irodalomjegyzék

Ian Sommerville: Szoftverrendszerek fejlesztése, Panem Könyvkiadó Kft., Budapest, 2007

A rendszerváltás technológiája, órai jegyzet, 2007/08-as tanév

http://en.wikipedia.org/wiki/Spiral_model

http://en.wikipedia.org/wiki/Internet_Information_Services

http://hu.wikipedia.org/wiki/MySQL

MySQL kézikönyv (http://dev.mysql.com/doc/)

http://hu.wikipedia.org/wiki/HTML

HTML dokumentáció (http://www.w3.org/TR/html401/)

http://hu.wikipedia.org/wiki/PHP

PHP kézikönyv (http://www.php.net/docs.php)

http://hu.wikipedia.org/wiki/JavaScript

http://hu.wikipedia.org/wiki/CSS

http://hu.wikipedia.org/wiki/Captcha

11. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani:

- Témavezetőmnek, Dr. Kuki Attilának szakmai támogatásáért, javaslataiért és türelméért.
- Kovács Tibornak, aki segített az alkalmazás grafikai munkáiban.