SZAKDOLGOZAT

Titkó Szabolcs

Debrecen
2009.

Debreceni Egyetem
Informatikai Kar

Diódakatalógus a weben

Témavezető:

Dr Kuki Attila
Egyetemi Adjunktus

Készítette:

Titkó Szabolcs
Mérnök Informatikus

Debrecen
2009.

TARTALOMJEGYZÉK

1. Bevezetés

1.1 Az Internet terjedése

Az internet a múlt század végén még lassan, de már biztosan terjedt. A XXI. század első éveiben ez a folyamat rohamos ütemű térhódítássá nőtte ki magát. Ennek az lett az eredménye, hogy az utóbbi 10 évben elérhetővé vált szinte bárki számára. Míg száz éve problémának számított egy zárt helyiség mesterséges megvilágítása is, mára már egyre ritkább az olyan háztartás, ahol ne lenne bekötve valamilyen szolgáltató nyújtotta internetkapcsolat. Az emberek közötti szórásos és pont-pont közötti kommunikáció szintén egy fontos igény, amelyre számos megoldás született az elmúlt évszázadokban. A távíró, a telefon, a rádió és a televízió után az internet nyújtja az eddigi legjobb megoldást az igényeink szerint jól definiált információszerzésre.

Az Internet elterjedésével együtt elszaporodtak a különböző weboldalak is. Sok közülük személyes oldal, hobbiból jött létre. Ezek az oldalak általában csak a készítőjük érdeklődési körét mutatják, persze ezzel másokat is a témára terelve. Vagy a design komolyabb, mint a logikai felépítés, vagy a funkcionalitás elenyésző. Persze van, aki pont ezeket látogatja, mert ez érdekli. Engem többek között az elektronika is érdekel amatőr szinten. Mivel minden nincs - szerintem nem is lehet - a fejemben, sokszor az interneten böngészve keresem a megoldást az elektronikával kapcsolatos problémáimra, vagy egyszerűen csak böngészek az érdekességek között. Ilyenkor többnyire áramkör- vagy nyomtatott áramkörtervező programokkal ismerkedem, esetleg elektronikai leírások után kutatok.

1.2 Témaválasztás

Néhány oldal hiánya többször is feltűnt, amikor ebben a témában böngésztem. Esetenként szükségem lett volna egy diódakatalógusra, tranzisztorkatalógusra vagy IC-katalógusra, ahol nem csak beszkennelt lapok több órás lapozgatásával és átkutatásával találom meg a keresett adatokat, hanem célirányos kereséssel kiszűrve a számomra éppen érdektelen tartalmakat, kiválogatva a lehetőségek közül a megfelelőt. Találtam

dokumentumokat, képeket, de adatbázisba foglalt műszaki jellemzőknek nyoma sem volt. Vajon hányan lehetnek még, akik ugyanezt hiányolják? Ezt nem tudnám megválaszolni, mivel napjainkban egyre kevesebben érdeklődnek az elektronika iránt áramköri alkatrész szinten. A javítás szinte teljesen kiment a divatból egyrészt a távol keletről érkező elektronikai termékek elérhető árai miatt, másrészt talán az emberi lustaság is jobban érvényesül, mint régebben. Ettől függetlenül biztos vagyok benne, hogy vannak olyanok, akik érdekeltek lennének a témában. Szerintem soha nem fognak teljesen kihalni azok a „barkácsoló-típusú" emberek, akik akár a spórolás céljából, vagy akár a kihívás kedvéért, de bele mernek nyúlni egy elektronikus készülékbe, mielőtt azt kidobnák a többi szemét közé. Emellett vannak és lesznek olyanok is, akik tanulási célból, esetleg szabadidős tevékenység keretein belül terveznek, majd építenek, vagy egyszerűen csak elkészítenek bizonyos elektronikákat.

Ebből kiindulva terveztem el, hogy létrehozok egy weblapot, amely egy online diódakatalógus szerepét vállalja először csak „kicsiben" az egyszerűség kedvéért. A tárhelyen egy többtáblás adatbázisból lehet a diódák műszaki jellemzőit, vagy magát a típust böngészni különböző szempontok alapján. Ha az adatbázis konkrét, korrekt és aktuális adattartalommal van feltöltve, akkor bizonyos idő elteltével megtalálják, felfigyelnek rá, és a látogatottságból kiderül, hogy csak nekem hiányzott ez a tartalom a világhálóról, vagy tényleg van értelme tovább fejleszteni és időt szánni arra, hogy a téma iránt érdeklődők mindig jelen időben beszéljenek erről a weblapról. Egyébként, ha nem lesz érdeklődés, akkor sem veszítettem semmit, hiszen mindig is érdekelt, hogy milyen módon lehet egy adatbázis köré működőképes webes kezelőfelületet építeni.

2. Alkalmazott technológiák és eszközök

2.1 HTML

A HTML a *HyperText Markup Language* rövidítése. Magyarul annyit jelent, hogy Hipertext jelölő nyelv. Az ilyen formátumú dokumentumok megtekintése egy úgynevezett „World Wide Web böngészőprogrammal" lehetséges. Ez azért van így, mert a HTML dokumentum-formátum a hyper-text egyik megvalósulási formája. Egy olyan szövegfájlról beszélünk, amely a szövegen kívül tartalmaz formázóutasításokat és a megjelenítendő objektumokra történő hivatkozásokat is.

A HTML eredeti célja az volt, hogy a dokumentumok szokásos, sorban egymás után olvasása helyett az egész dokumentum áttekinthetőbb, olvashatóbb legyen a szövegben található kapcsolatok, hivatkozások segítségével. A nyelv alkalmas dokumentumon belüli és dokumentumok közötti kapcsolatok létrehozására, mely kapcsolatokat az olvasó kezelhet. Itt ne a klasszikus értelemben vett dokumentumra gondoljunk, hanem inkább objektumokra, mint szöveg, grafika, zene, film. A szöveg olvasása közben a kereszthivatkozások segítségével könnyedén át lehet ugrani más területekre, aztán akár vissza is térhetünk az otthagyott objektumhoz. Ilyen szerkezetű például a Microsoft Windows súgója is.

A HTML eredeti verzióját a *World Wide Web Consortium* adta ki és a hivatalos szabványt ma is ez a szervezet írja tovább. A HTML utasítások köre folyamatosan bővül, a nyelv fejlődik, a szabványosítás pedig szép lassan követi, így nem minden böngésző tudja az összes utasítást értelmezni. Ha egy böngésző számára ismeretlen utasítással találkozik, akkor azt egyszerűen kihagyja, így az újabb keletű utasítások sem okoznak problémát a régebbi böngészőknek. Másrészről előfordul, hogy egyes böngészők másképp értelmeznek bizonyos utasításokat. Ezek miatt megeshet, hogy két böngésző ugyanazt a tartalmat másképpen jeleníti meg. A szabványosítás körüli eltérések ellenére a HTML nagymértékben megközelíti a platformfüggetlenséget.

2.2 JavaScript

A JavaScript, ahogy a neve is sugallja, a Java nyelv script változata. Erőssége leginkább abban rejlik, hogy rendelkezik a Java nyelv szinte összes lehetőségével, miközben szkript volta miatt forrásszinten be lehet ágyazni a HTML oldalakba. Segítségével egy interaktív weblap készítése közben nem kell teljesen különválasztani az oldalnak és az interaktivitást megvalósító interpretációnak a megírását, így lényegesen megkönnyíti a fejlesztéseket. Többek között emiatt van az, hogy több hasonló nyelv is született már (pl.: JScript, VBScript, mind az M$-tól), de ez idáig a JavaScript a legelterjedtebb, tegyük hozzá, nem véletlenül.

Egy fontos különbséget ne feledjünk. A JavaScript nyelvet, amit a Netscape fejlesztett ki, semmiképpen nem keverhetjük össze a Java nyelvvel, amit a SUN Microsystems alkotott meg. A JavaScriptet a Java alapján, annak esetenkénti kiváltására fejlesztették ki, tehát a névbeli hasonlóság nem a véletlen műve. Az elnevezés mégis megtévesztő, a sok különbség miatt, egyes szélsőséges vélemények szerint a két nyelvben csupán az első négy betű egyforma.

A Java nyelv megjelenésével utat nyitott a web oldalak interaktivitása felé, azonban ez az egyszerű szöveges HTML oldalakhoz képest egy hatalmas nagy ugrás. Úgy is fogalmazhatunk, hogy a JavaScript egy közbülső állomás a HTML és a Java között. Magyarán, ha szeretnénk egy kicsit élőbbé, interaktívabbá tenni az oldalainkat, akkor nem kell kapásból a Java nyelvet használni, mert az sok esetben körülményes. Ezzel ellentétben a JavaScript nyelv sok olyan egyszerűen elérhető szolgáltatást nyújt, amit a hétköznapi felhasználó is könnyen beilleszthet a dokumentumaiba. Annál is inkább, mert a JavaScript is hasonlóan egyszerű szöveges információ, mint a HTML, valamint interpretált, és nem végrehajtott nyelv, mint mondjuk a Pascal, vagy a C. Fontos még továbbá, hogy a JavaScript teljesen biztonságosnak nevezhető, tehát nem kell aggódnunk, hogy a böngészőben engedélyezett JavaScript mellett veszélyben lennének az adataink.

2.3 PHP

A PHP a dinamikus és interaktív weboldalak létrehozásának legegyszerűbb és talán leghatékonyabb eszköze. Rendkívül sokoldalú nyelv, gyakorlatilag minden olyan eszköz rendelkezésre áll benne, ami egy hatékony programnyelvhez szükséges. A PHP egy nyílt forráskódú, objektum-orientált, gyengén típusos programozási nyelv. Bő támogatottsággal rendelkezik és az elterjedtebb operációs rendszerek bármelyikén fut, a legtöbb kiszolgálóprogrammal és adatbázis-kezelőkkel együttműködve. Tehát akár fejleszthetünk UNIX rendszerre és minden probléma nélkül áttérhetünk NT alapokra. Ezt nevezzük hordozhatóságnak.

A PHP nem más, mint egy szerveroldali szkript nyelv, amely a legalapvetőbb feladatoktól az egészen összetett alkalmazásokig szinte mindenre képes, ami ebben a környezetben elvárható. Jellegzetessége, hogy a szkriptek a HTML kódba vannak beágyazva. A szerveren a PHP értelmező felismeri ezeket, lefuttatja és az eredményt a böngészőn keresztül adja vissza. Így egy weboldal PHP forrását böngészőnkből nem tudjuk megnézni, mert ott csak a PHP kód kimenetét találjuk. Bár a nyelv nem adatbázis orientált, hanem a C általános programozási nyelvre épül, mégis tartalmaz adatbázis kapcsolódási kiegészítő modult, amely lehetővé teszi, hogy elérhessük és a kódból generált HTML lapon megjelenítsük a legkülönbözőbb adatbázisok tartalmát is.

2.4 SQL

Az SQL egy szabványos nyelv, amely interfészt, kapcsolatot biztosít az adatbázishoz. Az adatbázis fogalmát a hozzá rendelhető legfőbb tulajdonságok megadásával tudnám leírni. Nem találtam egységesen elfogadott definíciót az adatbázis fogalmára, és bár ellentmondást sem találtam, minden definícióban más-más aspektust hangsúlyoztak.

Ez a kezelőfelület lényegesen eltér a hagyományos rekord-orientált, ciklusokat és elágazásokat tartalmazó lekérdező felületektől. Nem tartalmaz felhasználói képernyőkezelésre, vagy normál fájlkezelésre vonatkozó utasításokat sem. Emiatt mondhatjuk, hogy az SQL nem algoritmikus nyelv. Az adatkezelésnél az SQL a relációkat halmazokon végzi, és akár egész relációkra vonatkozó műveletek adhatók ki. A relációs

algebrán alapuló kezelőfelületet *SEQUEL*-nek nevezték el, utalva néhány alapvonására. A *SEQUEL* rövidítés a *Structured English Query Language* kifejezésre utal, ami annyit jelent, hogy egy strukturált, angol nyelvre épülő lekérdező nyelvről beszélünk. A parancsok kulcsszavai értelmes, a művelet jelentéséhez közel álló angol szavak, melyekből a parancsok mondatszerűen hozhatók létre. A lekérdezés szó azért szerepel a rövidítésben, mert bár a nyelv többre is alkalmas, az igazi ereje a lekérdezési részben, a relációs algebrára épülő komponensben rejlik. Az elnevezés később változott SQL-re, de a nyelv felépítésében, működési filozófiájában változatlan maradt.

2.5 Apache

Egy teljesen ingyenes nyílt forráskódú webkiszolgáló, melynek nagy szerepe volt a *World Wide Web* elterjedésében. A cél egy olyan webszerver program létrehozása volt, amely megfelel az Internet gyorsan változó követelményeinek, egyúttal biztonságos és üzleti célú felhasználásra is alkalmas. Karbantartása és fejlesztése várhatóan még sokáig biztosított lesz. Statikus és dinamikus weblapok közzétételére egyaránt használják, a világ internetes web-szervereinek nagy százalékán ez fut. Akár egy hagyományos PC-re is telepíthető, hiszen így fejlesztés közben a saját gépünkön tesztelhetjük a PHP kódunkat. Széles körben elterjedt, ismert, bevált, tesztelt, mindemellett nagymértékben személyre szabható és megfelelő beállításokkal elég biztonságos tud lenni.

Sok szabványt támogat, melyeknek nagy része fordított modulok formájában áll rendelkezésre a mag kiegészítéseként. Ezek a modulok sok területet lefednek a kiszolgáló oldali programnyelvtámogatástól kezdve a hitelesítési sémákig. Nagyon sok erőforrást takaríthatunk meg azzal, ha csak a számunkra szükséges modulokat töltjük be a szerverprogram indulásakor.

Sok webalkalmazást az Apache által nyújtott környezethez és szolgáltatásokhoz terveznek. Az Apache alkotja a webszerver komponenst a népszerű LAMP alkalmazáscsomagban, melynek további komponensei a MySQL adatbázisszerver és a PHP/Perl/Python programozási nyelvek.

3. Alapok

3.1 HTML oldalak készítése

Amennyiben a HTML oldalakat a legegyszerűbb eszközzel akarjuk fejleszteni, akkor elég megnyitni a Jegyzettömböt, bár ez több okból sem célszerű, helyette egy komolyabb szerkesztő például a NotePad+ már elég jól használható. Egyébként bármelyik olyan szövegszerkesztő használható, ami képes egyszerű szövegfájlba menteni (pl.: VI, EMACS, Ms-DOS Editor, stb.).

Ugyanakkor előismeret nélkül ezekkel az eszközökkel szinte esélytelen honlapot készíteni. Ebben az esetben ajánlhatóak az ún. *WYSIWYG (What You See Is What You Get)* típusú színes honlapkészítői környezetek. Előnyük, hogy első ránézésre semmilyen programozói tudást nem igényelnek és könnyű ezeket használni. Hátrányuk, hogy sablonosak, alacsonyabb interaktivitás jellemzi őket, és bár adnak lehetőséget a kódszintű szerkesztésre, erősen korlátozzák azt.

Léteznek még más típusú szerkesztő programok is a fenti kettő mellett, amelyek úgy adnak támogatást, hogy a folyamatot a fejlesztő és nem a program vezérli. Elsőre hagyományos karakteres szerkesztőnek tűnhetnek, ám nagyon hasznos eszközökkel támogatják a HTML dokumentum megírását. Ilyenek a struktúra szerinti kódszínezés, a HTML elem beillesztése és a szintaktika ellenőrzés. Az ilyen programok megfelelő programozói előismerettel hatékonyak és gyorsak.

3.2 Webtárhely

A webtárhelyet valamilyen internetes szolgáltatónál találhatjuk meg. A szolgáltató egy webszerverének az erőforrásait felosztják több felhasználó között, ezt virtuális webtárhely szolgáltatásnak nevezzük. A felhasználók a rendszer által dedikált tárhelyet kapnak, melyeknek a nyilvános tartalmát egy egyedi domén néven érhetik el. A nagyszámú felhasználó kézben tartható és hatékony kezelése érdekében a webtárhely szolgáltatás tartalmaz egy adminisztrációs felületet. A kisebb forgalmú weboldalakat bérelt webtárhelyen

üzemeltetni költséghatékony. A nagyobb szolgáltatóknak vannak fizetős és ingyenes lehetőségei. Ha fizetünk a szolgáltatásért, akkor nagyobb tárhelyet kaphatunk és az oldalunk reklámmentes lesz, így több felületet használhatunk ki, ráadásul kevesebb felugró ablakra számíthatunk.

Osztott tárhelyszolgáltatók az esetek többségében az egyes szolgáltatásoknak fizikailag külön kiszolgáló rendszereket tartanak fenn. Az ügyfélkiszolgáló és adminisztrációs rendszer, a levelező kiszolgáló, az adatbázis szerver, a webszerver funkciók külön fizikai kiszolgálókon működnek. A legtöbb webkiszolgáló alacsony költségű Linux vagy FreeBSD operációs rendszer alapú LAMP szerver. A LAMP a Linux Apache MySQL PHP rövidítése, tehát minden szolgáltatás, amire egy webszervernek szüksége lehet.

A felhasználó számára elérhető technológiák csoportját az határozza meg leginkább, hogy mely operációs rendszerre van építve a szolgáltatás. A M$ Windows alapú webhosting esetében például ASP.NET és Microsoft SQL Server de akár PHP és MySQL Server támogatást is választhatunk, amíg LAMP szerver esetén csak PHP szkriptnyelvű oldalakat készíthetünk MySQL Server támogatással. Tegyük hozzá, az interneten található magánoldalak számára az esetek többségében ez is bőven elegendő.

3.3 HTTP protokoll

„A HTTP (HyperText Transfer Protocol) egy információátviteli protokoll a világhálón. Az eredeti célja a HTML lapok publikálása és fogadása volt. A HTTP fejlesztését a World Wide Web Consortium és az Internet Engineering Task Force koordinálta RFC-k formájában. A legfontosabb RFC az 1999-ben kiadott RFC 261622, amely a HTTP/1.1 verziót definiálja. Jelenleg ez a legelterjedtebb verzió.
A HTTP egy **kérés-válasz alapú protokoll** kliensek és szerverek között. A kommunikációt mindig **a kliens kezdeményezi**. A HTTP klienseket gyűjtőnéven user agentnek is nevezik. A user agent jellemzően, de nem feltétlenül webböngésző.
A HTTP általában a TCP/IP réteg felett helyezkedik el, de nem függ tőle.” [1]

[1] Nagy Gusztáv: Web programozás I.

3.4 Relációs adatmodell

A relációs adatmodell legfontosabb eleme a matematikai reláció fogalma. A logikai adatbázis kereteit határozza meg. A megvalósítás részletei eltűnnek a felhasználó elől. Nem foglalkozik a fizikai adattárolással, a memóriaműveletekkel. Ahogy a többi adatmodell, ez is tartalmaz statikus és dinamikus elemeket is. A statikus oldalon definiálja azokat a jellemző adatszerkezeteket, struktúrákat, amelyekkel dolgozni fog. A dinamikus oldalon definiálja a struktúrákon értelmezett műveleteket, operációkat. A struktúra és a műveletek mellett a relációs adatmodellben megjelenik még az adatintegritási komponens, mely sem a strukturális, sem a műveleti komponensnek nem része. Az integritási feltételek az adatok értékei közötti kapcsolatokat szabályozzák. A relációs adatmodell így az alábbi három komponens együttese:

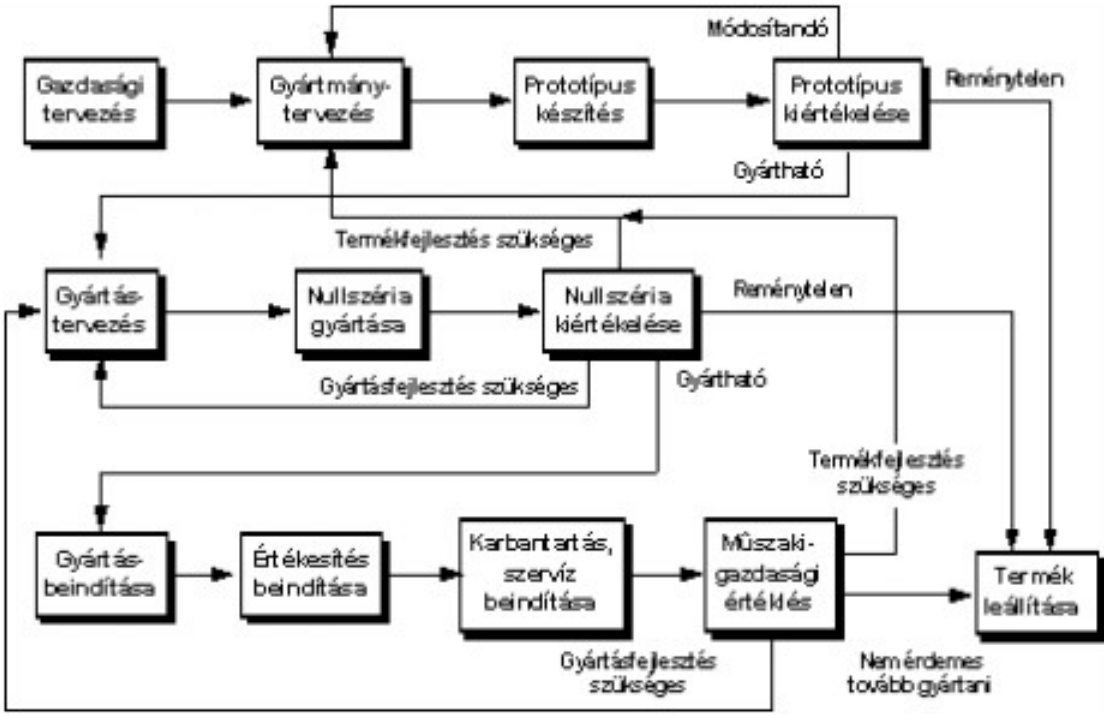
- Adatstruktúra
- Műveletek
- Integritási feltételek

A relációs adatmodell napjaink legelterjedtebb adatmodellje. Alapjait 1970-ben definiálta E. F. Codd amerikai kutató. Cikkében bizonyítani akarta, hogy az akkor elterjedő hálós adatmodell mellett van más alternatíva is. A modell gyakorlati alkalmazása csak az 1980-as években vált általánossá. Az a lényege, hogy az egyedeket, tulajdonságokat és kapcsolatokat egyaránt táblázatok, úgynevezett adattáblák segítségével adja meg.

A relációs adatmodell strukturális része egyszerű és könnyen érthető. Ez az egyszerűség nem csak a közérthetőség miatt fontos, hanem az alkalmazhatóságot is növeli. Olyan műveleti rész csatlakozik a modellhez, ami egyszerűbb kezelői felületet biztosít a programozási nyelveknél, hogy minél szélesebb körben lehessen használni. Az adatmodell integritási feltételeinek előnye, hogy a szabályok magában az adatbázisban kerülnek letárolásra, így a relációs adatbázis kezelő rendszer csak olyan tevékenységeket enged, melyek nem sértenek semmilyen integritási előírást.

4. Szoftverfejlesztés alapok

Minden terméknek van egy élettörténete (életciklusa). Ez akkor kezdődik, amikor felmerül az igény a termékkel kapcsolatban, és addig tart, amíg végleg ki nem lesz vonva a forgalomból. Akkor a legjellemzőbb az életciklus elnevezés, amikor a terméknek rendre újabb, továbbfejlesztett változatát hozzák létre. Ilyenkor a fejlődés minden állomásán meg kell ismétlődnie a tervező, kísérletező, tevékenységnek, amely a termékkel kapcsolatos újabb és újabb igények megfogalmazásától egészen a gyártás megindításáig tart. Tehát a fejlesztés egy termék életciklusának az a szakasza, amelyik a vele kapcsolatos igény felmerülésétől a gyártás megindításáig tart. A fejlesztés szakaszában egyrészt a terméket, másrészt a technológiát kell megtervezni és tesztsorozatokkal igazolni. A termék és a technológia természetesen hatással van egymásra.



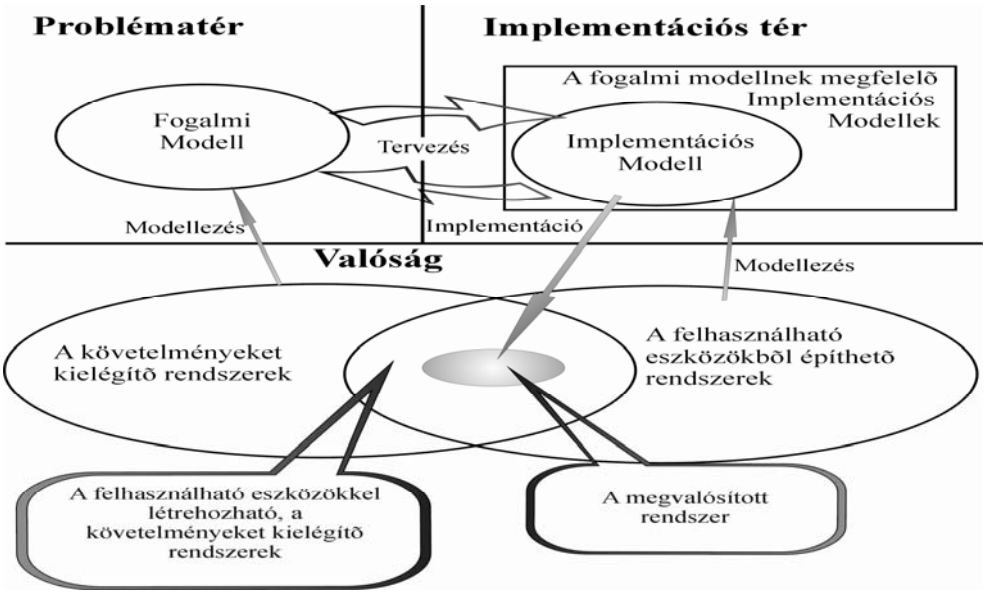
1. ábra - Egy elképzelés az életciklusról

Olyan programot kell tervezni, amely a felhasználónak szimpatikus. A használathoz szükséges információkat mindenképpen tartalmaznia kell, továbbá a képernyőn keresztüli adatbevitelnél megfelelő tájékoztatószövegek is szükségesek. A boldog felhasználó minden szempontból pozitív hatással van a fejlesztőre.

4.1 Alapproblémák

A fejlesztés folyamatát három jellegzetes lépésre bonthatjuk: modellezés, tervezés, implementáció. A **2. ábra** alapján értelmezhetjük a három tevékenységet és láthatjuk, hogy ezek mennyire függenek egymástól, és hogy mennyire szükségesek.

A fejlesztés jelentős részét a modellezés teszi ki. A modellnek nem az a célja, hogy tükrözze a valóság minden apró részletét. Inkább azokra a tulajdonságokra kell koncentrálni, amelyek az alkalmazás szempontjából fontosak. Így ugyanarról a valós dologról több modell is alkotható, vagy fordítva, egy modell több valós dolognak is megfelelhet.



2. ábra - Modellezés

Az **2. ábra** három „világot” mutat. Az alsó rész a valóságot jelöli. A bal felső térrész az úgynevezett problématér, ahol a speciális feladathoz kötődő dolgok vannak. Ebben a térben alakul ki a létrehozandó rendszer fogalmi modellje. A jobb felső térrész az úgynevezett implementációs tér, amelyet a fejlesztő szempontjából a megvalósításhoz felhasználható eszközök határoznak meg. Itt található az implementációs modell, ami a megvalósítandó rendszert írja le.

A fejlesztés kezdetekor általában a fogalmi modell egy változatából indulunk ki, és a követelményeket kielégítő valóságos rendszerhez kell eljutnunk. Ami a legtöbb gondot okozza, hogy a rendszerek általában bonyolultak, amiből számos probléma származik. A legfontosabb, hogy a bonyolult modellek áttekinthetetlenek, megértésük, kezelésük nehézkes. Minél bonyolultabb a rendszer, annál több a hibalehetőség, ezért a bonyolultságon uralkodni kell.

4.2 A bonyolultság feletti uralom

A bonyolultságot akkor vehetjük észre, amikor annyi mindent kellene fejben tartani, amennyit már nem tudunk. Különböző részletekre koncentrálunk, és végül elveszünk közöttük. Olyan modellt kell alkotni, ahol az aktuális részletek fejben tartható mennyiségűek és tervezés közben egy körre kell koncentrálni. Erre két eszköz áll rendelkezésre, a részletek eltakarása (absztrakció), illetve a probléma egyszerűbb részekre bontása (dekompozíció).

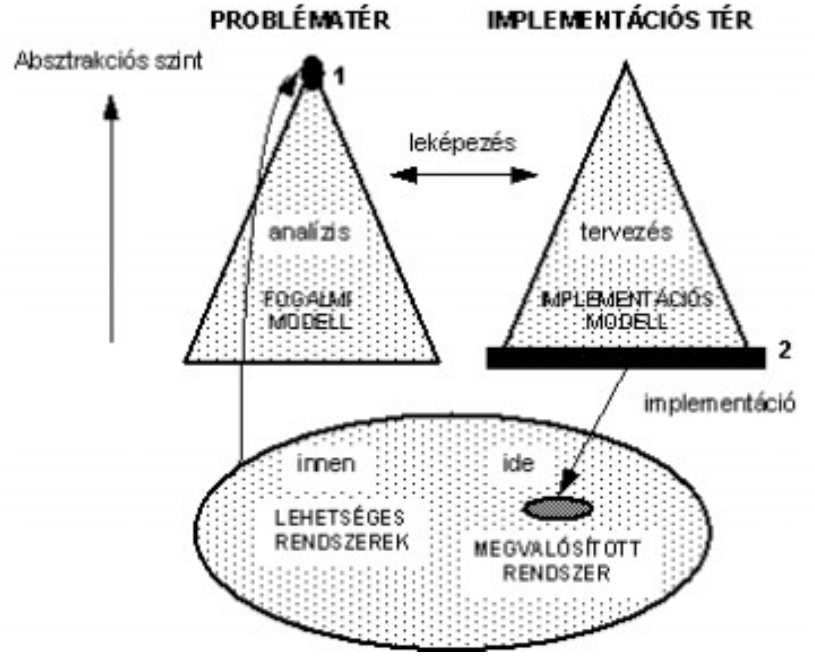
Az absztrakcióval eltakarjuk a szükségtelen, zavaró részleteket így létrehozva egy kevésbé bonyolult modellt, ami sokkal átláthatóbb, így könnyebben kezelhető. Minél összetettebb, bonyolultabb dolgot tekintünk eleminek, annál magasabb szintű az absztrakció. Ha fordítva vesszük, akkor ahogy egyre jobban közeledünk az apró részletekhez, annál konkrétabbak vagyunk.

Ha a különböző feladatokat különböző egységek látják el, mint egy vállalat osztályai, akkor egy átlátható struktúrát kapunk. Ezeket külön-külön is kezelhetjük, mint önálló egységeket. Itt jön be a képbe a dekompozíció. A dekompozíció alatt azt értjük, amikor egy rendszert egyszerűbb részrendszerekre bontunk, melyek együttműködve az eredeti rendszernek megfelelően viselkednek. Ilyenkor részrendszereket definiálunk és meghatározzuk ezek működési módját. Így ezeket egymástól függetlenül lehet tovább fejleszteni.

Az absztrakció és a dekompozíció nem csak a programozásban jelenik meg, ezek az ember ösztönös gondolkodási technikái, létezésük megfigyelhető a beszélt nyelvben is. Mindkét technikát alkalmazhatjuk a fogalmi és az implementációs modell kialakításakor egyaránt.

4.3 A fejlesztés folyamata

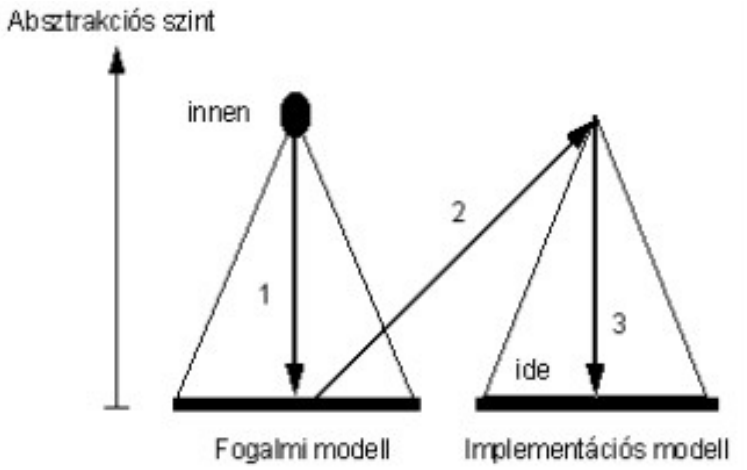
Egy szoftver fejlesztése általában azzal kezdődik, hogy a valóságban felmerül valamilyen probléma vagy hiány, melynek a megoldása vagy pótlása lehetséges és célszerű lehet valamilyen informatikai rendszer segítségével. Mivel a valóságot a fogalmi modell alapján figyeljük meg, annak csúcspontja lehet a kiindulási pontunk. **(3. ábra)** Ilyenkor természetesen még csak a rendszer körvonalai jelennek meg, igen magas absztrakciós szinten. Az esetek többségében a folyamat abból áll, hogy az 1-es pontból (problématér csúcsa) kell eljutni a 2-es pontba (implementációs tér alja), közben megalkotva mindkét modellt. Ez az út többféleképpen bejárható.



3. ábra - Modellalkotás

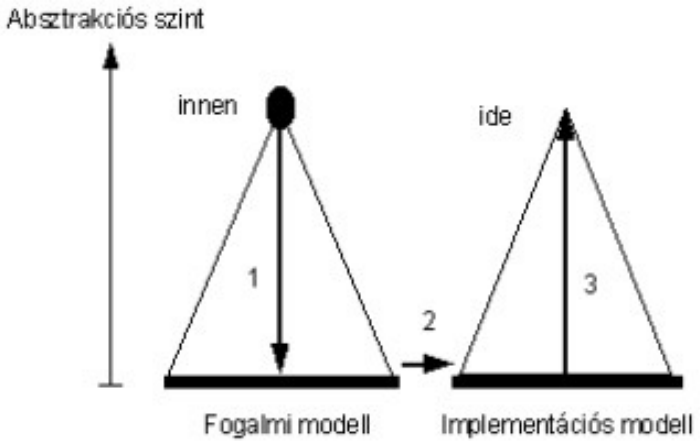
A felülről lefelé (top-down) haladás **(4. ábra)** esetében először teljes egészében ki kell alakítani a fogalmi modellt úgy, hogy az absztrakciós szintjét fokozatosan csökkentjük. Az absztrakciós szint csökkentését úgy érjük el, hogy eldöntjük, hogy az addig nem részletezett fogalmakat, tulajdonságokat és műveleteket milyen egyszerűbb komponensekből lehet felépíteni. Tehát a legmagasabb absztrakciós szintről döntések sorozatával jutunk el a konkrétumok szintjére. Minden döntésnél ki kell választani valamit, amiről döntünk. Ez a domináns fogalom, más néven a strukturáló objektum.

A **4. ábra** alapján először finomítással kialakítjuk a teljes fogalmi modellünket. Ezután az implementációs modell esetében ugyanezt a stratégiát alkalmazzuk. Azonban az implementációs modell finomítása közben nem tekinthetünk el a már kész fogalmi modelltől, hiszen a konkrét implementációnak ugyanúgy kell viselkednie, mint a konkrét fogalmi modellnek.



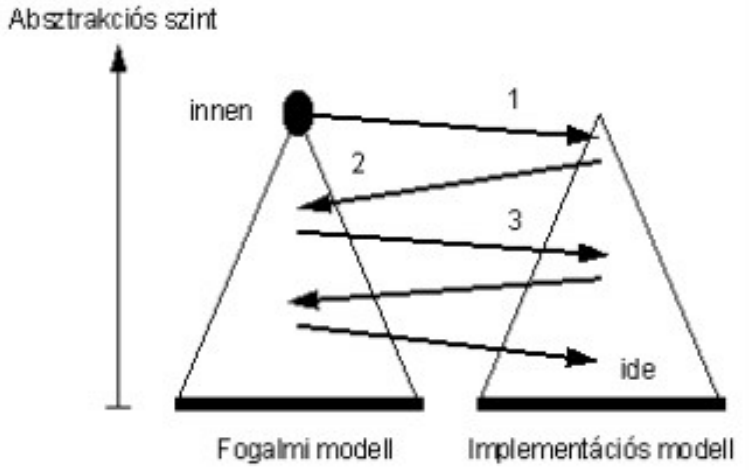
4. ábra Top-Down haladás

Egy másik lehetőséget mutat be a **5. ábra**. Itt az első szakaszban ugyanúgy járunk el, mint az előző esetben. Ezután azonban az implementációs modellt alulról felfelé (bottom-up) hozzuk létre. A második szakaszban a fogalmi modell konkrétumai alapján összeállítjuk az implementációs modell konkrétumait (szintézis), majd az implementáció könnyebb kezelhetősége érdekében magasabb absztrakciós szintű fogalmakat alkotunk.



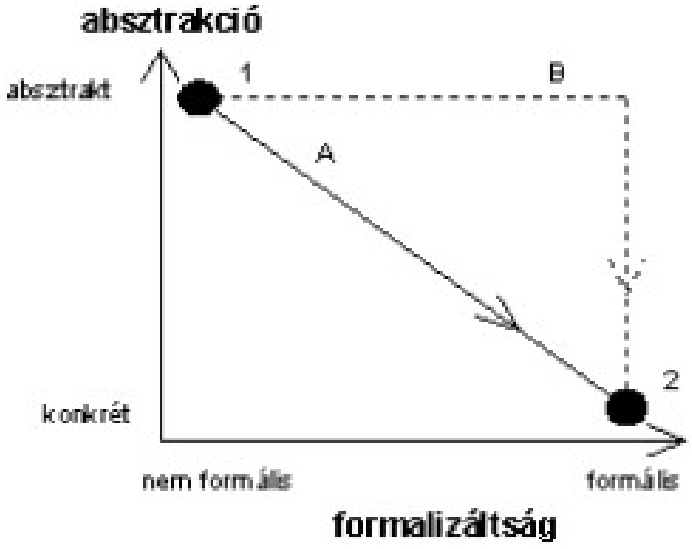
5. ábra – Bottom-Up haladás

Számos módszertan javasolja az **6. ábra** szerinti stratégiát, miszerint kis analizálás után jöhet egy kis tervezés, majd ezt ismételgetjük. Tehát miután végrehajtottunk egy finomítási lépést a fogalmi modellen, átgondoljuk a következményeket, majd ennek figyelembevételével végrehajtunk egy finomítási lépést az implementációs modellen is. Ezeket a lépéseket ismételgetve egy olyan végállapothoz jutunk, ahol a fogalmi- és az implementációs modell szerkezetileg fedik egymást. Az előző két stratégiánál ez nem teljesül minden esetben. Itt a problématér fogalmai biztosan felismerhetőek lesznek az implementációban, továbbá a szoftver könnyebben megérthető, és tudjuk, hogy hol kell módosítani, ha szükségessé válna.



6. ábra – Vegyes módszer

Akár a problématérben, akár az implementációs térben vagyunk, a modellek és leírásaik vizsgálata történhet más szempontból is: az absztrakció és a formalizáltság függvényében. **(7. ábra)** Az egyes pont egy magas absztrakciós szintű, egyáltalán nem formális leírás, ezt általában a felhasználó használja. Innen kell eljutni a kettes pontba, amely az implementációs térben szigorúan formalizált informatikai fogalmakat, magát a programszöveget jelenti.

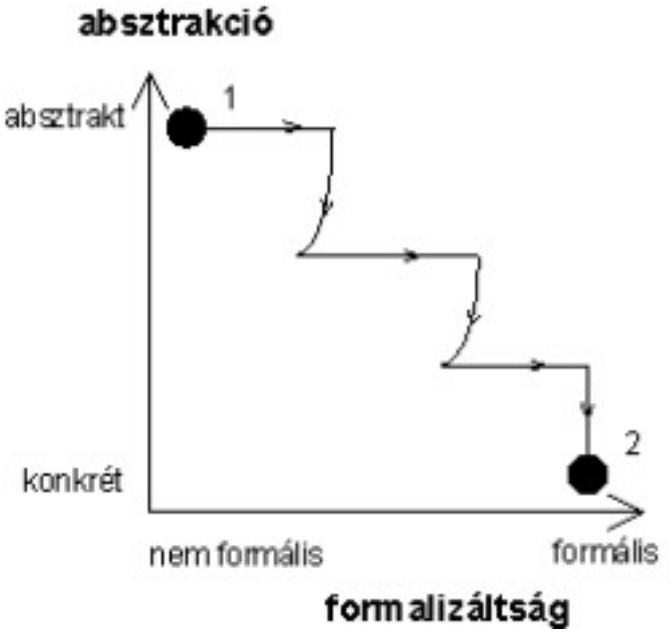


7. ábra

A formalizáltság növelését a gyakorlatban specifikálásnak nevezzük. Amikor csökkentjük az absztrakciós szintet, így egy magasabb absztrakciós szintű elemet alacsonyabb szintű elemekből állítunk össze, tervezésnek hívjuk. Ha ebben az értelemben vesszük a fogalmakat, akkor nem foglalkozunk vele, hogy melyik térben vagy modellben tevékenykedünk.

A **7. ábrán** az egyes pontból kell eljutni a kettes pontba. A két, nagybetűvel jelölt útvonal közül a B jelű lenne a legoptimálisabb. Itt a magas absztrakciós szintű fogalmak segítségével olyan szigorú formális leírást készítenénk, amely jól definiált transzformációval közvetlenül a kódhoz juttatna. Ez a megoldás csak a legegyszerűbb esetekben használható, mivel magas absztrakciós szintű bonyolultabb fogalmi modellről nem tudunk olyan szigorú formális leírást készíteni, amely ésszerű terjedelemben leírható lenne.

Az A jelű úton való haladás azt jelentené, hogy egy időben specifikálunk és tervezünk. Ha jobban belegondolunk, rá kell jönnünk, hogy ez emberileg senkitől sem várható el, ezért a specifikációs és tervezési lépések a gyakorlatban egymást váltogatják. **(8. ábra)** Az utolsó tervezési lépés, a kódkészítés kivételével a tervezés nem csak az absztrakció csökkenését jelenti, mivel egy magasabb absztrakciós szintű elemet konkrétabbakból építünk fel. Ezen a szinten a terv már kevésbé formális, további specifikációt igényel.



8. ábra

5. Tervezés

A specifikáció első lépése a felhasználói igények felmérése, a megrendelő azon igényeinek pontos rögzítése, dokumentálása, melyekből a leendő rendszer funkcionalitását meg lehet határozni. A konkrét megvalósítási kérdésekre itt még nem szabad kitérni, bármilyen késztetést is érez az ember. Ha az igényfelmérést a konkrét megvalósítás szem előtt tartásával alakítjuk ki, akkor nem tudjuk elkerülni, hogy rendszertervezői döntéseket hozzunk. Ezek a döntések nem ebbe a fázisba valók, ráadásul vannak esetek, amikor ezen döntések meghozatala kívül esik a mi hatáskörünkből, vagy felelősségkörünkből. Ez a helyzet akkor állhat elő, amikor egy nagy, összetett szoftver fejlesztése folyik, ami több embert vesz igénybe. Ilyenkor mindenki végzi a saját feladatát, és azt megfelelően dokumentálja a többiek számára, ezzel biztosítva és elősegítve a fejlesztők közötti szükséges és elegendő formális kommunikációt.

5.1 Megvalósíthatósági elemzés

A megvalósíthatósági elemzés felméri azt, hogy a javasolt rendszer ténylegesen megfelel-e a megrendelő követelményeinek, illetve hogy üzletileg indokolt-e egy ilyen rendszer kifejlesztése. A rendszer technikai megvalósíthatósága helyett az elemzések napjainkban már inkább arra koncentrálnak, hogy az üzleti, vagy működési célok elérését mennyire fogja segíteni. A megvalósíthatósági elemzés lehet akár egy lehetőségekre vagy problémákra vonatkozó elemzés következménye is. Ez határozza meg a kezdeti felhasználói követelményeket és az esetleges alternatívákat is. Vannak meghatározott feladatok és bizonyos feladatokat az igények határoznak meg. Az igényelt környezetet csak annyira kell leírni, hogy lehetővé váljon a probléma-megfogalmazás kialakítása és a megvalósíthatósági alternatívák azonosítása.

Egy olyan taktikai tevékenységnek, mint a megvalósíthatósági elemzés szükséges előzménye a stratégiai tervezés. Ez többek között kifejezi a programozóval szemben támasztott elvárásokat. Kisebb volumenű rendszereknél nincs feltétlenül szükség stratégiai tervezésre, ilyenkor a megvalósíthatóság elemzése során kell kitérni rá.

A megvalósíthatósági elemzés a célok rövid megfogalmazásával kezdődik, becslést kell végezni a javasolt rendszer kiterjedésével és bonyolultságával kapcsolatosan, és el kell készíteni az elemzésre vonatkozó terveket. Ezután kell megfogalmazni a problémát. Ebben a lépésben kell meghatározni a rendszer információigényeit, a megoldandó problémáit és a szükséges szolgáltatásait. Itt még el kell kerülni a részletes leírást. Az igényelt környezet leírásához lehet adatfolyam-ábrákat készíteni, ezzel megkönnyítve a folyamatok leírását. Ezután a fontos teljesítmény-tényezőket kell azonosítani, ezzel meghatározva a kritikus folyamatokat. A funkcionális követelményekhez tartozó nem-funkcionális követelményeket szintén fel kell venni a követelményjegyzékbe. Ezek lehetnek:

- adathozzáférési korlátozások
- auditálás és ellenőrzés
- általános korlátok
- megfigyelés
- biztonság

5.2 Szolgáltatások meghatározása

A szolgáltatásokat, melyeket az általunk fejlesztett termék produkálni fog, általában a megrendelő határozza meg. Itt derül ki, hogy a rendszernek egyáltalán mi lesz a feladata, és hogy milyen formában kell majd működnie. A megrendelőnek is van szerepe a fejlesztésben, méghozzá egy (vagy több) szakértő kijelölése, akinek rálátása van a megoldandó feladatra. Neki tudja a fejlesztő feltenni a problémával kapcsolatos szakmai kérdéseket, később pedig ő tudja tesztelni az elkészült részeredményeket. Ő fogja meghatározni a konkrét feladatokat. Természetesen a szakértőnek is kell, hogy legyen valamennyi rálátása az informatikára, hiszen sokszor olyan jellegű technikai kérdésekben is neki kell dönteni, amelyeknek az adott feladathoz kevesebb köze van. A termék elkészülte után a szakértő lesz a megrendelő oldalán, aki a rendszert nem csak kezelni tudja, hanem érti is.

5.3 Követelmény meghatározás

Nem „csak” egy terméket kell létrehozni, hanem megoldást kell nyújtani egy problémára. Az informatikai megoldás azt jelenti, hogy a rendszer minden szereplője a munkájához szükséges információt a megfelelő helyen, a megfelelő formában és a kellő időben megkapja. A követelmények meghatározása során funkcionális és nem-funkcionális követelményeket kell leírni a rendszerről. Ebbe bele kell érteni az összes korlátot és jövőbeli kiterjesztést is. A lehető legpontosabban meg kell határozni a rendszer mennyiségi jellemzőit, a teljesítményre, rugalmasságra és biztonsági szintjére vonatkozó elvárásokat. Ügyelni kell arra, hogy ne tartalmazzon szükségtelen megszorításokat, amelyek felesleges plusz költségekkel járnak. Egymásnak ellentmondó követelmények szintén nem fordulhatnak elő, hiszen nem csak sok bonyodalmat okozhatnak, de a rendszer helyes működését is erősen veszélyeztetik.

A megfigyelés, a dokumentumok elemzése, a felhasználók kikérdezése a jelenlegi környezet felmérésében segíthet. Minél többet tud az elemző a felhasználók igényeiről, annál inkább tud ő is követelményeket javasolni. Ebben az esetben természetesen a felhasználókat is meg kell kérdezni a javasolt követelménnyel kapcsolatban, hiszen a követelmények leginkább rájuk tartoznak. Fel kell ismerni, hogy mit igényelnek és miért, hiszen a leendő

rendszert már a tervezés során is ismerni kell. Fel kell készülni a váratlan helyzetekre is, át kell venni az összes „ha ez-és-ez történik, akkor hogyan reagáljon a rendszer” – típusú kérdést.

A funkcionális követelmények leírják, hogy mit kell tudnia a rendszernek. A legtöbbször használati esetekre van bontva, és minden esetre tartalmazza a rendszer összes funkciójának a működését. Például aktualizálások, lekérdezések, adatok, jelentések, stb. A nem-funkcionális követelmények azt írják le, hogy hogyan, vagy milyen minőségben kell egy lehetőséget nyújtania a rendszernek. Vonatkozhatnak a rendszer egészére, vagy egyes részeire.
Ilyenek:

- **Szolgáltatási szintekre vonatkozó követelmények**
  - működési időszak (hétvége, ünnepnap stb.)
  - rendelkezésre állás (a működési időszak százalékában)
  - válaszidők
- **Adathozzáférési korlátozások**
  - védelmet igénylő adatok
  - olvasás vagy módosítás korlátozása bizonyos felhasználói szerepkörökre
- **Biztonság**
  - mentések gyakorisága
  - rendszer összeomlás (kézi rendszer, csökkentett rendszer, tartalék rendszer a visszaállításig)
- **Megfigyelés**
  - jelentések tartalma, gyakorisága
  - kihasználtsági szintek figyelése
- **Auditálás és ellenőrzés**
  - rendszer-auditálás (fontos tranzakciók nyomon követése)
  - teljesítmény-auditálás
- **Korlátozások**
  - kapcsolat más rendszerekkel
  - ember-számítógép kapcsolat követelményei
  - archiválás

5.4 Logikai rendszerterv

A logikai rendszerterv két részletben tevődik össze, az általános logikai tervből és a részletes logikai tervből. Az általános logikai rendszerterv nagy vonalakban rögzíti a rendszer architektúráját, a főbb modulokat. Ebben a szakaszban már nincs szükség az ügyfél részéről aktív részvételre, azonban nagyon hasznos, ha a fentebb említett szakértő is áttekinti a terveket. Bizonyos kereteken belül ilyenkor még lehet módosítani az igényeken és annak megfelelően a logikai terveken. A részletes logikai rendszerterv már tartalmazza az összes fontos modult, a programfolyamatot, interfészt.

A logikai rendszertervezés folyamán részletesen meg kell fogalmazni a követelmény-specifikációban megfogalmazott feldolgozási szerkezeteket. Ezután meg kell határozni a hatékony működéshez szükséges felhasználói felületeket, dialógusok formájában. Itt határozzuk meg a dialógusok szerkezetét, a menü- és parancsszerkezeteket a dialóguson belüli illetve a dialógusok közötti navigációs követelményekkel együtt. Ebben a szakaszban kell elkészíteni a lekérdező funkciók és a módosító funkciók lekérdezési elemeinek logikai specifikációját is.

5.5 Tesztelés

A szoftverek minőségének biztosítása egy fejlesztés során egyre nagyobb szerepet kap, főleg az egyre növekvő komplexitásra való tekintettel. Ennek kulcsfontosságú eleme a tesztelés. A korszerű fejlesztési eszközök és módszerek használata ellenére is elkerülhetetlen, hogy a szoftverbe hibák kerüljenek. A szoftver biztonságos használata érdekében a hibák döntő többségének felderítése és kijavítása elengedhetetlen.

A tesztelés nem más, mint a program használata szimulált környezetben, közben megfigyelés, dokumentálás. A környezet kialakítása közben szem előtt kell tartani, hogy olyan környezetet alakítsunk ki, amely a szoftver használatának lehetőség szerint minden szélsőséges esetére kiterjedjen. A tesztelés során tapasztalt minden rendellenes működést dokumentálni kell.

A szoftverek működési jellemzőit különböző szempontok alapján lehet csoportosítani. A tesztelés szintén ezen szempontok alapján történik:

- **Funkcionális tesztelés**
  - **Installációs teszt**
    A rendszer telepítésének tesztelése különböző eshetőségek (különböző hardver-szoftver konfigurációk, feltételek) esetén.
  - **Általános funkcionális teszt**
    A rendszer működésének vizsgálata normál működés esetén. A teszt során ellenőrizzük, hogy a rendszer funkciói az elvártnak megfelelően működnek, a teszt során a kívánt eredményeket kapjuk.
  - **Szélsőérték funkcionális teszt**
    A rendszer működésének vizsgálata szélső bemeneti/kimeneti értékek esetén. A teszt során ellenőrizzük, hogy a rendszer funkciói az elvártnak megfelelően működnek, a teszt során a kívánt eredményeket kapjuk.
  - **Konfigurációs teszt**
    A rendszer funkcionalitásának tesztelése eltérő hardver/szoftver feltételek mellett.
  - **Mennyiségi teszt**
    A rendszer funkcionalitásának tesztelése nagy mennyiségű bemenő, kimenő, ill. adatbázis adat esetén.
  - **Biztonsági teszt**
    A szoftver jogosultsági rendszerének tesztelése ellenőrzi, hogy a rendszer adataihoz csak az elvárt felhasználók férnek-e hozzá.

- **Teljesítménytesztelés**
    - **Általános teljesítményteszt**
      A rendszer működésének sebességét összevetjük az elvárt értékekkel. A teszt segítségével fényt deríthetünk a rendszer sebesség szempontjából kritikus pontjaira, szűk keresztmetszeteire.
    - **Referencia teljesítményteszt**
      A rendszer működésének sebességét egy másik, a tesztelt rendszerhez hasonló funkcionalitású rendszer sebességével vetjük össze.
    - **Terheléses teljesítményteszt**
      A rendszer sebességének vizsgálata különböző munkaterhelések esetén. A teszt során rögzítjük a terhelés mértékét (pl. felhasználók, tranzakciók száma), valamint a rendszer működési sebességét az adott terhelés alatt.
    - **Konkurencia teljesítményteszt**
      A rendszer teljesítményének vizsgálata abban az esetben, ha a rendszernek többszörösen kell ugyanahhoz az erőforráshoz (pl. ugyanahhoz az adatrekordhoz) hozzáférnie.
    - **Sokk teljesítményteszt**
      A rendszer működésének vizsgálata extrém körülmények között. Ezek az extrém körülmények lehetnek a tervezettnél nagyobb terhelések, kevés memória/erőforrások, hardver problémák, áramszünet, stb.

5.6 Szoftvervalidáció

Célja, hogy megmutassa, a rendszer egyezik saját specifikációjával, és hogy a rendszer megfelel a megrendelő elvárásainak, vagyis a rendszert jól készítettük el. Ez leginkább a megrendelő megnyugtatására való eszköz, persze a fejlesztő szempontjából is nyújt bizonyos védelmet. A validációnál két fő technikát használnak: a szoftverátvizsgálásokat és a tesztelést. Mivel a validáció költséges folyamat, a tervezését már a fejlesztési folyamat elején el kell kezdeni.

5.7 Szoftver evolúció

A nagy és összetett rendszerek hosszú élettartamúak, közben pedig sokat változhatnak. Egyrészt az eredeti rendszerben felmerült hibákat kell kijavítani, másrészt az újonnan felmerült követelmények alapján kell beleépíteni az újabb funkciókat, elemeket. Közben a rendszer működési környezete is megváltozhat, sőt, a rendszer felhasználói köre is bővülhet, cserélődhet. Mivel a cégek jelentős része is szinte teljesen függ a szoftverrendszerétől, beláthatjuk, hogy a szoftverevolúcióra komolyan oda kell figyelni.

5.8 Üzemeltetési dokumentáció

Az üzemeltetési dokumentáció írja le, hogy mik a szoftver rendszerkövetelményei, hogyan működtessük és telepítsük a szoftvert az igényeinknek megfelelően. A lehetséges konfigurálási lehetőségeket és az alapbeállításokat is bele kell foglalni, hogy a szoftver testre szabása ne okozzon problémát a felhasználónak. Bár az lenne az ideális, ha erre nem lenne szükség, de a lehetséges hibajelenségek elhárítását is tartalmaznia kell. Ezek nem feltétlenül a szoftver hibái, lehetnek például különböző kompatibilitási problémák olyan esetekben, amikor a szoftver működéséhez szükséges feltételek csak részben teljesülnek. Ilyenkor megtörténhet, hogy a szoftver működik, de bizonyos funkciók használata rendszerhibához vezet. Szükség van még egy részletes felhasználói dokumentációra, és mindezeken kívül arra, amit a megrendelő esetleg még beleálmodott leendő rendszerének leírásába.

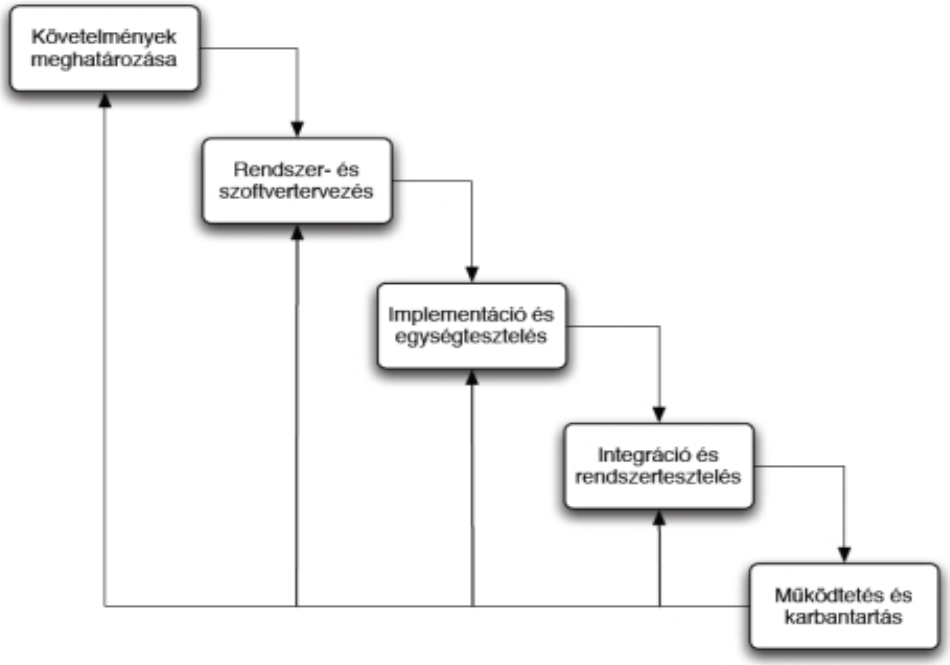
6. Szoftverfejlesztési modellek

A szoftverfejlesztés az esetek többségében olyan problémákkal szembesíti a fejlesztőket, amelyek megoldása a fejlesztési folyamat elején nem belátható. Olyan probléma megoldási folyamat ez, melynek során kompromisszumok sorozatán keresztül számos szereplő elképzeléseit kell összehangolni. Ezek a körülmények egy közös nyelvezetet igényelnek, amely a szempontrendszereket képes szisztematikusan közvetíteni a megvalósítók felé.

A szoftverfejlesztések számának növekedésével megjelent az igény a fejlesztés folyamatának racionalizálására. Vannak olyan közös szoftverfejlesztési modellek, amelyek adnak egy iránymutatást a termék életciklusának meghatározásához. Azért nagyon értékesek ezek a modellek, mert rengeteget takarítanak meg a fejlesztők számára abból az időből, amit a munkaterv elkészítésével töltenének. Lássunk néhányat.

6.1 Vízesés modell

A vízesés modell volt az első publikált rendszerfejlesztési modell. A gyakorlatban széles körben használatos, mert egy olyan alapot szolgáltat, amely bármelyik feladatnál használható. Az életciklust fázisokra bontja, így a fejlesztésben felmerülő tevékenységeket egymás utáni, jól elválasztható lépésekben írja elő. Addig nem indulhat el a következő fázis, ameddig az előző be nem fejeződött. A **9. ábrán** látható az elnevezés oka: a folyamat fázisonként halad lépcsőzetesen a megvalósulásig. Először megértjük a megoldással szemben támasztott követelményeket, majd megtervezzük a megoldást, építünk és tesztelünk, majd alkalmazzuk a megoldást. Egyszerű és könnyen megérthető. Az ábrán is látható, hogy van lehetőség visszacsatolásokra, korrekciókra a szintek között, de a modell alapvetően lineárisan írja le a fejlesztési folyamatot. Az egyes fázisok végén természetesen vannak döntési pontok, melyeken ellenőrizzük eddigi munkánkat és eldöntjük, hogy haladhatunk-e tovább a következő fázisban.



9. ábra – Vízesés modell

1. **Követelmények meghatározása**

   A funkcionális és nem funkcionális követelmények, korlátok, továbbfejlesztési lehetőségek leírása. A leendő felhasználókkal történt konzultációk alapján meg kell határozni a rendszerrel szemben támasztott elvárásokat.

2. **Rendszer- és szoftvertervezés**

   Itt kerül rögzítésre először nagy vonalakban, majd részletesebben a rendszer architektúrája, főbb moduljai. Meg kell határozni a követelmény-specifikációban megfogalmazott feldolgozási szerkezeteket.

3. **Implementáció és egységtesztelés**

   A rendszert összehasonlítjuk a saját specifikációjával és megvizsgáljuk, hogy sikerült-e megfeleltetni a rendszert a megrendelő elvárásainak.

4. **Integráció és rendszertesztelés**

   Ebben a lépésben történik a különálló részrendszerek összeintegrálása, majd az így összeállt rendszer tesztelése.

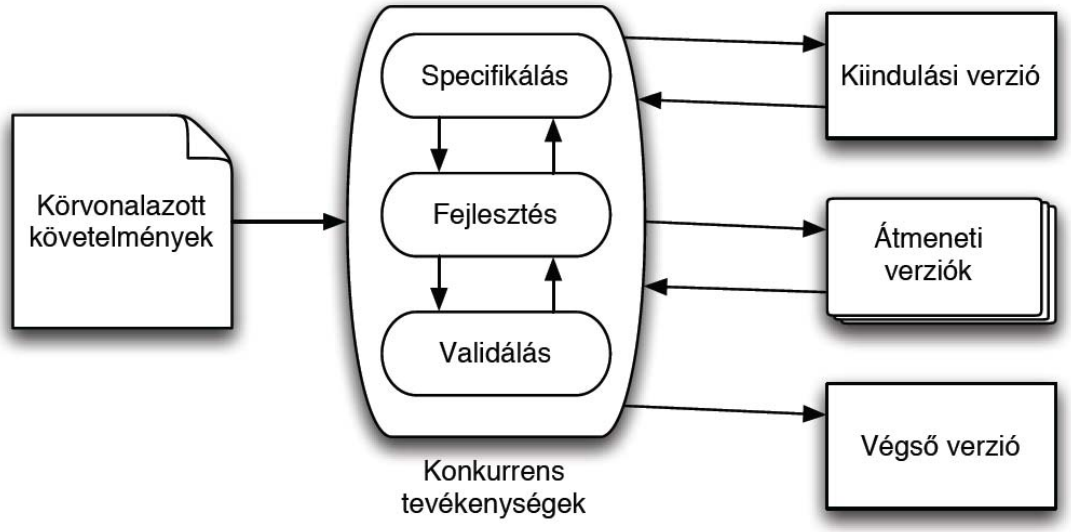
5. **Működtetés és karbantartás**

   A rendszer életének optimális esetben a leghosszabb szakasza. Elkezdődik az éles üzemeltetés és biztosítani kell eleinte ennek felügyeletét, mert hiába az összes tesztelés, mindig ez az egyik sorsdöntő pont. Persze, ha valamilyen hiba merülne fel, akkor annak kijavítása (pl.: biztonsági- és hibajavító frissítések) is ebbe a szakaszba tevődik.

Mára a vízesés modell elavulttá vált. Előnye, hogy fázisonként készül egy részletes dokumentáció, amely végül egy átgondolt, stabil rendszert eredményez. A fejlesztési feladatok viszont fogós problémát jelentenek és a probléma felmérése kiterjed a teljes fejlesztési folyamatra, tehát a fejlesztés különböző lépéseit nem választhatjuk el egymástól.

6.2 Evolúciós modell

A vízesés modell problémáira az egyik megoldást a prototípussal dolgozó evolúciós modell jelentette. Itt először ki kell dolgozni a szoftver prototípusát, melyet a megrendelő kipróbál, és az ő véleményezése alapján kerül további finomításra. Ezt a tevékenységet ciklikusan addig folytatják, amíg elő nem áll a kész rendszer. Az evolúciós modell kétféleképpen valósítható meg.



10. ábra - Evolúciós modell

Az egyik megvalósítási mód a feltáró fejlesztés. A megrendelővel közösen történő követelmény feltárás után egyből következik a kész szoftver elkészítése. A megrendelő a kész szoftvert látva megfogalmazza a további igényeket, követelményeket. Így áll elő a minden igényt kielégítő végtermék.

A másik lehetőség az eldobható prototípusok készítése. Ezeknek az a célja, hogy először teljesen megértsük a megrendelő igényeit, majd ezekre alapozva készíteni egy konkrétabb specifikációt, ami alapján ki lehet alakítani a kész rendszert. Tehát az eldobható modellek a tökéletes feladat meghatározást veszik célba.

Természetesen ennek a módszernek is megvannak a hátrányai. A rendszer szerkezete nem lesz igazán átlátható a folytonos módosítások hatására. Nem lehet biztosítani a megfelelő dokumentációt, így a rendszer karbantartása nagyon nehézkes lesz. Rövid élettartamú kis és közepes rendszerek fejlesztéséhez ideális.

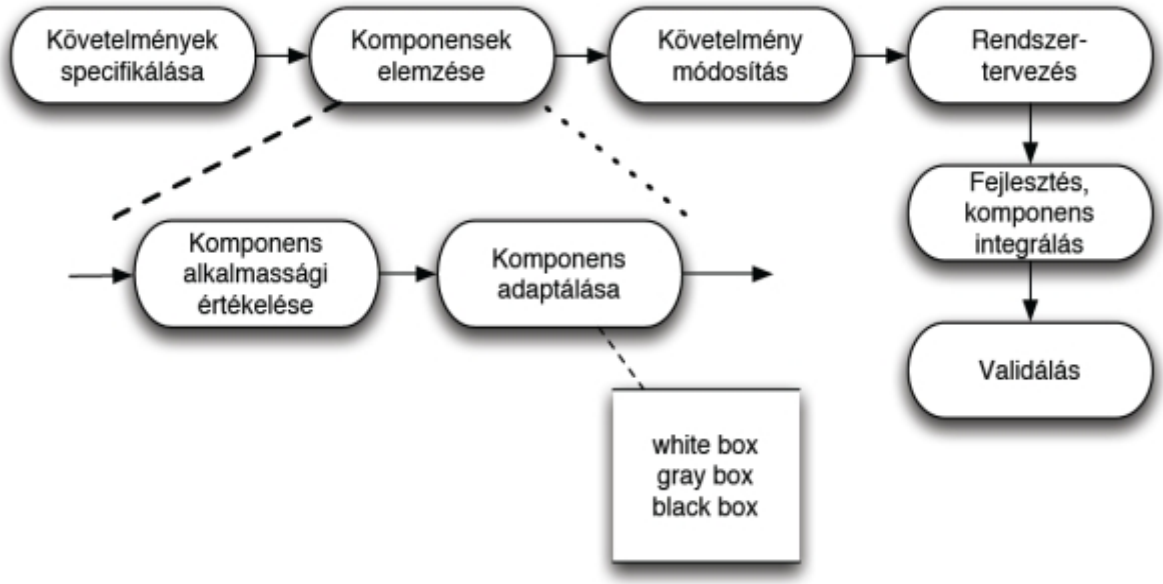
A vízesés modell és az evolúciós modell kombinálható egy nagyobb modellben. A vízesés modell a specifikáció egyértelmű részein, az evolúciós modell eldobható prototípusai a nem egyértelmű részeken, a feltáró fejlesztési mód pedig a felhasználói felület megvalósításában használható.

6.3 Formális rendszerfejlesztési modell

Kapcsolódik a vízesés modellhez, de a fejlesztési folyamat alapja a rendszerspecifikáció futtatható programmá történő transzformálása formális matematikai eszközökkel. Legismertebb formája a Cleanroom-módszer, melyet az IBM fejlesztett ki. Minden egyes fejlesztési szakasz után formális módszerekkel bebizonyítják annak hibátlanságát, így a tesztelés elmarad. A formális rendszerfejlesztésről nem mondhatjuk el, hogy széles körben használatos, mert speciális szakértőket igényel, és nem költséghatékonyabb más módszereknél. Szigorú biztonságot, megbízhatóságot és védelmet igénylő termékeknél érdemes használni.

6.4 Újrafelhasználás-orientált fejlesztési modell

Ez a modell nagymértékben az elérhető újrafelhasználható szoftverkomponensekre támaszkodik. Persze sok esetben szükséges némi átalakítás, de még így is nagymértékben felgyorsítja a fejlesztést, hiszen lecsökkenti a kifejlesztendő részek számát, a költségeket és a kockázatot. Az újrafelhasználás-orientált modell egyik ismert példája a komponensorientált szoftverfejlesztés. Ebben a megvalósítandó szoftverkomponensek száma minimalizálható, ám a meglévő komponensek az esetek többségében nem felelnek meg az elvárásoknak, elkerülhetetlen a kompromisszumok sora.



11. ábra – Újrafelhasználás-orientált fejlesztési modell

A **11. ábra** alapján a modell lépései a következők:

1. **Követelmények specifikálása:** A funkcionális és nem funkcionális követelmények, korlátok, továbbfejlesztési lehetőségek leírása.
2. **Komponensek elemzése:**

   Az adott követelményspecifikáció alapján meg kell keresni, hogy melyek azok a kész komponensek, amelyek megvalósítják azt. Az esetek többségében az illeszkedés nem tökéletes, és a kiválasztott komponens csak a funkciók egy részét fedi le.

3. **Követelmény módosítás:**

   A megtalált komponensek alapján a követelményeket elemezni kell, majd módosítani az elérhető komponenseknek megfelelően. Ahol ez nem lehetséges, ott újra a komponenselemzés jön, és az új megoldás keresése.

4. **Rendszertervezés (újrafelhasználással):**

   Meg kell tervezni a rendszer szerkezetét, vagy felhasználni egy létező vázat annak figyelembevételével, hogy milyen újrafelhasznált komponensek lesznek integrálva. Úgy kell mindezt összehozni, hogy a komponensek tudjanak együttműködni.

5. **Fejlesztés és integráció:**

   A nem megvásárolható komponenseket ki kell fejleszteni, és a kereskedelemben kapható rendszerekkel össze kell kapcsolni. A rendszerintegráció itt sokkal inkább a fejlesztési folyamat része, mint különálló tevékenység.

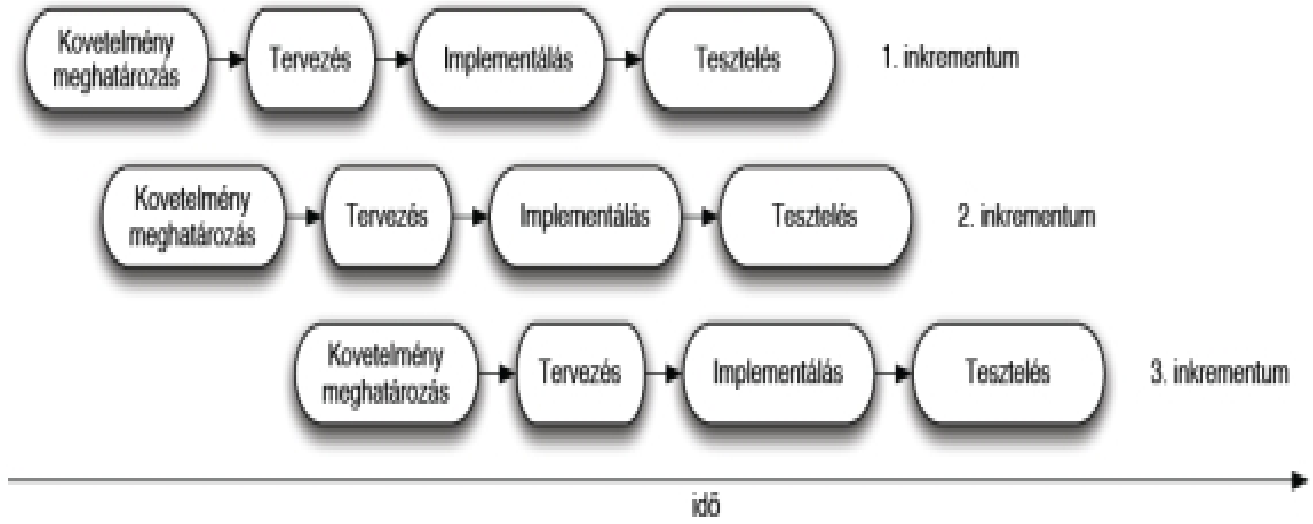
6. **Validálás:** az 5.6 pontban már említett módon

6.5 Inkrementális szoftverfejlesztési modell

A vízesés modell korlátai akkor váltak láthatóvá, amikor kiderült, hogy a szoftver változási hajlama még az egyedi rendszerek esetén sem szorítható merev fázishatárok közé. Még a leggondosabb tervezés ellenére is szinte minden rendszer átadásakor felmerülnek olyan módosítási igények, amelyek visszahatnak még a követelményekre is. Az evolúciós modell egy rosszul strukturált és nehezen karbantartható rendszert eredményez. Az inkrementális szoftverfejlesztési modell a két módszer előnyeit próbálja összekovácsolni.

A modell szerint egy felületes specifikáció során megfogalmazott szolgáltatásokat úgynevezett inkremensekbe soroljuk. Ezeknek az inkremenseknek a funkcionalitását pontosan kell specifikálni. Érdemes kis méretűre választani, de egy rendszerfunkciót célszerű egyetlen inkremensbe sorolni. Az inkremensek fejlesztése egymás után történik, amint elkészül egy, azonnal integrálva lesz a teljes rendszerbe, így a megrendelő már tesztelheti is. Ha a prioritási sorban előrébb álló inkremensek kerülnek korábban megvalósításra, természetesen azok

tesztelésére több időt kell szánni. Az inkremensek az integrálás előtt egymástól függetlenek, így akár más-más szoftverfejlesztési modell alapján is fejleszthetőek.

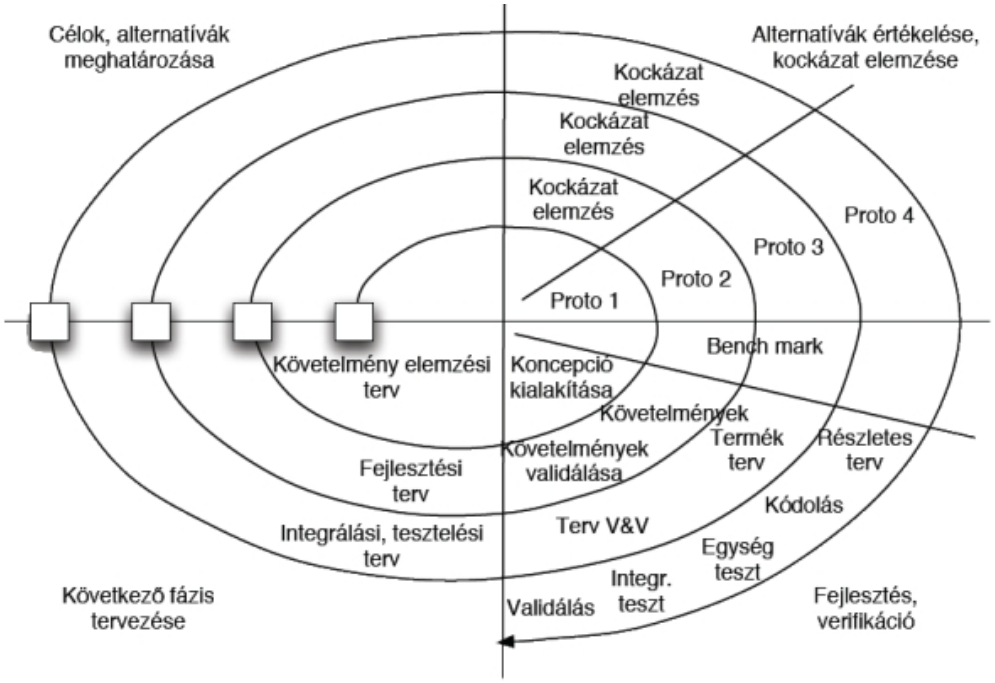


12. ábra - Inkrementális szoftverfejlesztési modell

6.6 Spirális szoftverfejlesztési modell

A szoftverfolyamatot nem tevékenységek és a közöttük található esetleges visszalépések sorozataként tekinti, hanem egy spirálként. A spirál minden egyes körben a szoftverfolyamat egy-egy fázisát reprezentálja. A legbelső kör foglalkozik a megvalósíthatósággal, a következő a rendszer követelményeinek meghatározásával, aztán a rendszertervetéssel, stb. A spirál mindegyik ciklusa négy szektorra van felosztva:

- célok és alternatívák meghatározása
- kockázat becslése és csökkentése
- a fázis termékének megvalósítása és validálása
- következő fázis tervezése



13. ábra - Spirális szoftverfejlesztési modell

A spirális szoftverfejlesztési modell fókuszában a kockázatelemzés van, aminek keretében a projekt minden szakasza a kockázat minimalizálás elérésének van alárendelve. A modell előnyei, hogy a tevékenységek struktúrája egyértelmű, egyszerű a megvalósítás és biztos alapot nyújt a tervezési fázisok feladatainak egységes szemléletű végrehajtásához. A fejlesztési munkát alapozhatjuk a saját elképzeléseinkre, a tervezési fázisban az általunk elképzelt rendszer megvalósíthatósága és a felhasználók igényeinek maradéktalan kielégítése a cél.

A spirális modell esetében a kiindulási pont a követelmények meghatározása és a kockázatok specifikálása. Ezután következik a fejlesztési tervek kidolgozása, a költségbecslések és a megvalósíthatósági alternatívák elkészítése. Az első körben kidolgozott tervek lesznek a felhasználók által is minősíthető prototípusok fejlesztésének, majd a programtervezési és tesztelési munkák elvégzésének alapjai. A prototípusok készítése nagy mértékben csökkenti a kutatási- és fejlesztési tevékenységek bizonytalanságát, mert olyan folyamatok végrehajtására kerül sor, amelyekben a felhasználói elképzelések és a rendszerről feltárt ismeretek alapján kidolgozhatók a kívánt modellek. Ezek a prototípusok úgy készülnek, hogy be lehessen mutatni a fejlesztés szempontjából kritikus komponensnek

minősített eredményeket, folyamatokat, bonyolult algoritmusokat, ember-gép kommunikációkat, amelyeket nem lehet előre megfogalmazni egyértelmű igényként.

A fejlesztési fázisban választhatunk fejlesztési modellt, megtehetjük, hogy akár minden spirálban más modellt alkalmazunk. Az adott spirálban használt fejlesztési modell típusát a kockázatanalízis eredménye alapján kell megválasztani. A következő spirálba történő továbblépés előtt elemezni kell az addig végrehajtott munkát. Az elemzés eredményezheti azt a helyzetet is, hogy a munkát nem kell folytatni, mert már megfeleltünk a követelményeknek. Ha mégis folytatni kell, akkor a következő spirálban újra kezdődik a folyamat, természetesen már magasabb szinten.

7. Webes alkalmazás ismertetése

A szakdolgozat további témája egy webalkalmazás, funkcióját tekintve konkrétan egy webes diódakatalógus. Jelen esetben konkrét megrendelőről nem beszélhetünk, inkább egy olyan csoportról, akiknek egy olyan adatbankra van szükségük, amely lehetőséget nyújt nekik a tervezői, javítási tevékenységük technikai adatokkal történő támogatására. Mindez folyamatos frissítéssel mindig naprakész állapotban tartva egyszerű lekérdezésekkel megvalósítva egy gyors és hatékony segítséget jelenthet a papír alapú katalógusokkal szemben.

A weblapot első körben csökkentett funkcionalitással és extrák nélkül tervezem, hiszen a létére vonatkozó igényt önmaga fogja felmérni. A látogatottság lesz az egyik mérvadó adat, az üzenőfalon hagyott felhasználói visszajelzések pedig remélhetőleg ennek az adatnak a megerősítése.

Az előző témakörben részletezett modellek között válogatva, előnyeiket, hátrányaikat, bonyolultságukat, a jelen feladathoz való alkalmazási lehetőségeiket mérlegelve az inkrementális szoftverfejlesztési modellt láttam a legalkalmasabbnak a feladathoz

7.1 Igényfelmérés és a követelmények meghatározása

Igazából, mint a bevezetésben is említettem, a katalógus hiányával kapcsolatos igények felmérése részben megtörtént, annyi különbséggel az általános igényfelmérésektől, hogy ebben az esetben egyszerűen magamból és sorstársaimból indultam ki. Tudom, hogy nem vagyok egyedül, csak abban nem vagyok biztos, hogy mekkora ez a tábor. A következőkben ismertetem a felhasználói követelményeket.

A rendszer célja, hogy különböző diódák tulajdonságait a típus alapján le tudjuk kérdezni, esetleg több típust összehasonlítani. A másik fő funkció lényege az, hogy tulajdonságok alapján lehessen megkeresni a megfelelő típust. Egy kommunikációs modul is szükséges, ahol a további igényeket lehet fölvetni a diódatípusokkal, tulajdonságokkal, vagy akár az alkalmazással kapcsolatban is.

A rendszer funkcionalitásával szemben támasztott követelmények:

1. A rendszer felhasználóit három különböző csoportra kell bontani. Ezek a csoportok a következők: látogatók, tagok, adminisztrátorok. A látogatók egyszerűen tudnak böngészni a lapon, tudják használni az adatbázist, el tudják olvasni a komplett weblap tartalmát. A tagok tudnak kommunikálni, tehát üzenetet írni, nyilvánossá tehetik az elérhetőségeiket egymás számára. Az adminisztrátor(ok) lesznek azok, akik az oldal működéséért, fenntartásáért felelősek. A tagok kérdéseikkel hozzájuk fordulhatnak.

   A tagoknak és az adminisztrátoroknak biztosítani kell egy megfelelő biztonsági szintű bejelentkezési lehetőséget, és biztosítani a további azonosításukat. A két csoportnak különböző jogosultságokkal kell rendelkeznie.

2. Biztosítani kell egy regisztrációs lehetőséget azoknak, akik tagok akarnak lenni. A későbbiekben tudniuk kell módosítani az adataikon, esetleg törölni a regisztrációjukat. A regisztráció közben a felhasználók saját maguk adják meg az adataikat. A felesleges kockázatok és bonyodalmak elkerülése végett ellenőrizni kell, hogy valós e-mail címmel történik a regisztráció.

3. A regisztrálás során kapott jelszó módosításának lehetőségét biztosítani kell a tagok számára. Érdemes a regisztráció után biztonsági okokból kötelezővé tenni a jelszómódosítást az első bejelentkezéskor.

4. Az elfelejtett jelszó helyett lehessen újat kapni. Az azonosítást ilyenkor az e-mail címmel lehet megvalósítani.

5. A regisztrációs folyamatban megadott adatokat két részre kell osztani. Nyilvános és privát. Ennek egyszerű oka, hogy mindenki csak annyi adatot lásson a másikról, amennyit az szeretne megmutatni magából.

6. A tagok nyilvános kommunikációját egy üzenőfal teszi lehetővé. Ezt minden regisztrált felhasználó láthatja és írhat is bele.

7. Az adminisztrátornak rendelkeznie kell jogosultságokkal arra, hogy módosítson a rendszeradatbázis tartalmán. Tudnia kell hozzáférni, módosítani és törölni a felhasználókat és adataikat.
   A többi adatbázishoz is hozzáférést kell biztosítani az adminisztrátoroknak, hiszen az adatbázisban lehetnek hibák, melyeket csak ők módosíthatnak. Továbbá az üzenőfal tartalmának időnkénti módosítása is szükséges lehet, ha esetleg valaki nem tartaná be a „netikett” szabályait, és oda nem illő szavakat használ a bejegyzései során.

8. Az adatbázisban történő keresésre mindenkinek legyen joga, nem kell regisztrációtól függenie.

9. A keresés eredményeit listában kell megjeleníteni, a részletek megjelenítéséhez a lista elemét ki kell választani.

10. Legyen egy általános leírás a félvezetőkről az oldalon, bemutatva ezzel a téma alapjait.

A rendszer funkcionális követelményei mellett bizonyos nem funkcionális követelményeket is le kell írni. Ezeket a későbbiek során nem kell inkremensekbe sorolni, viszont a tervezés során végig szem előtt kell tartani. A diódakatalógus nem funkcionális követelményei a következők:

1. Mint minden alkalmazásnál, itt is első szempont az egyértelmű, felhasználóbarát kezelőfelület. Amellett, hogy a kezelőfelület jól, könnyen használható, oda kell figyelni, hogy egy esztétikus látvány fogadja a látogatókat. Ezen szempontok alapján, következetes tervezéssel kialakítható egy könnyen használható dekoratív webalkalmazás.

2. Mivel az alkalmazás bizonyos részei regisztrációt igényelnek, oda kell figyelni a felhasználók személyes adatainak biztonságára. A bejelentkezési adatok és a tagok nem nyilvános adatai mások számára nem lehetnek elérhetőek.
   A felhasználói jelszavak tárolása nem történhet az adatbázisban direkt módon. Megfelelő adatszerkezetek tervezésével meg kell oldani, hogy a jelszóellenőrzés végrehajtása lenyomatok tárolásával valósuljon meg.

3. Webes alkalmazásról lévén szó, a felhasználók a legkülönfélébb böngészőkkel fogják látogatni a lapot. Ahogy a 2.1 pontban is írtam, a más-más böngészők másképpen értelmezhetnek bizonyos kódokat. Nincs értelme még megkísérelni sem egy weblapot minden böngészőre optimalizálni. Ehelyett a legelterjedtebb, leggyakrabban használt böngészőket kell célba venni. A megjelenítés egységességét még így sem lehet garantálni, de inkább legyenek kis vizuális különbségek, mint nem működő funkciók. Ha tartjuk magunkat az élő szabványokhoz, akkor sok kellemetlenséget elkerülhetünk.

4. Nem elhanyagolható szempont a bolondbiztosság sem. A programnak ellenőriznie kell a bevitt adatok helyességét. Nem túl elegáns, amikor egy rosszul beírt adat miatt valamilyen rendszerüzenet figyelmeztet minket mogorván, sokszor olyan hibára hivatkozva, aminek nyoma sincs a rendszerben, például, hogy az adatbázis sérült. (Ilyen esetben egyébként az „ismeretlen hiba” kifejezéssel találkozhatunk.) A programnak legyen saját kivételkezelője, ami figyelmeztet minket, hogy például szám

7.2 Az adatbázis

Mindenekelőtt az adatbázist kell megtervezni, hiszen a végterméknek ez lesz a lényege, a tartalmat és az egésznek az értelmét ez adja. Ha elkészül, köré lehet építeni a weblapot, mint egy kezelőfelületet. Először azt kell megvizsgálni, hogy milyen jellegű adatbázissal érdemes dolgozni. A tartalomtól függően lehet egy-, vagy többtáblás adatbázist használni. Ez véglegesen akkor derül ki, ha összegyűjtjük az összes adatot, amit tárolni akarunk. Igazából egy valós dolgot modellezünk le, ezért el kell döntenünk, hogy mennyire kell a valósághoz ragaszkodni. Ha valamilyen adatot nem fogunk felhasználni, akkor feleslegesen bonyolítja az adatszerkezetet és foglalja az értékes tárhelyet. Az adatszerkezettel szemben támasztott követelmény, hogy az összes tárolandó értéket egyértelműen tartalmazza minél kevesebb belső redundancia mellett. Miután összegyűjtöttük a számunkra értékes adatokat, szükség van a relációsémák formális vizsgálatára, amely a redundanciákat detektálja és az optimalizálást lehetővé teszi. Ezt normalizálásnak nevezzük.

Redundanciának nevezzük, ha például egy adat többször szerepel egy táblában. Ilyenkor módosítás esetén több helyen is el kell azt végezni. Ha új adatot kell bevinni, előfordulhat, hogy elvileg egyenlő értékű adatok mégsem egyeznek, vagy egyszerűen idővel feleslegessé válnak bizonyos rekordok. Törlésnél pedig akaratlanul fontos adatokat veszíthetünk a redundancia miatt.
Normalizáláskor az 1NF-re[2] hozás a relációs modellnél kötelező. A további normálformák egyre szigorúbb feltételeket írnak elő (2NF <= 3NF <= BCNF <=4NF), amelyek kiküszöbölik a redundanciát és az aktualizálási anomáliákat. Az ezek szerinti normalizálás célszerű, de nem kötelező. A gyakorlatban azt kell mérlegelni, hogy a redundancia és az anomáliák mennyire jelentenek súlyos veszélyt, indokolt-e azok megszüntetésével a táblák számát növelni.

[2] NF – NormálForma

7.2.1 Az adatbázis tervezése

Egyelőre nem törekedtem a teljességre, hiszen az első működőképes verzió még nem csak diódakatalógusként működik, hanem részben azt a célt is szolgálja, hogy kiderüljön, van-e rá egyáltalán igény. Ezen oknál fogva kezdetnek összesen a diódák négy fontos csoportját választottam ki. Ezek az általános felhasználású diódák, a varikap diódák, a sáv kapcsoló varikap diódák és az egyenirányító diódák.

Az adatok összegyűjtése után egyértelműen kiderült, hogy többtáblás megoldást igényel a feladat. A diódák fő csoportjai külön táblákban helyezkednek el. **(14. ábra)** Így a táblák között nincs szükség kapcsolatokra, hiszen ha célirányosan keresünk egy diódát, akkor tudnunk kell, hogy milyen csoportban kell azt keresni. Még ha tulajdonságok alapján keresünk, akkor sem fogjuk az egyenirányító diódák tulajdonságai között keresni a változtatható értékű diódák bizonyos értékű zárófeszültségét, hiszen ez a két dióda teljesen más funkciókat lát el.

14. ábra – Katalógus-adattáblák

Egy további táblára van még szükség, méghozzá a regisztrált felhasználók adatait tartalmazó táblára. A regisztrációs folyamatban itt kerül mentésre minden adat, amire a továbbiakban szükség lesz.

A katalógus egyetlen adatbázisban tárolja a diódák és a felhasználók adatait. Az adatbázisban szereplő táblák és funkcióik:

- user: regisztrált felhasználók és adataik tárolása
- varikap: változtatható kapacitású diódák osztálya
- sav_varikap: sáv kapcsoló varikap diódák osztálya
- alt_dioda: általános felhasználású diódák osztálya
- egyenir: egyenirányító diódák osztálya

7.2.2 Az adatbázis tábláinak bemutatása

Az egyszerűség kedvéért az elsődleges kulcsot a minden táblában ’id’-vel jelöltem. A táblák nevei utalnak a dióda-osztályokra, éppúgy, mint a mezőnevek a tulajdonságokra. Ez a későbbi karbantartásokat, módosításokat könnyíti meg.

A ’user’ tábla mezői:

- id: felhasználó azonosító
- vnev: vezetéknév
- knev: keresztnév
- mail: e-mail cím
- nick: felhasználói név
- pass: jelszó
- valid: megerősítő kód a regisztrációhoz
- rank: adminisztrátor és tag között tesz különbséget

A ’varikap’ tábla mezői:

- id: elsődleges kulcs
- tipus: dióda típusa, a pontos beazonosításhoz
- bekot: bekötés (melyik irányba vezet)
- labszam: dióda kivezetéseinek száma
- tok: tokozás típusa
- V_Rmax: záróirányú feszültség maximuma
- I_Fmax: nyitóáram maximuma
- C_dmin: dióda kapacitása előírt záróirányú feszültség mellett
- zarfesz: zárófeszültség
- kapar: kapacitás arány
- atV_1: minimum feszültség, ahol az eszköz még üzemszerűen használható
- atV_2: maximum feszültség, ahol az eszköz még üzemszerűen használható
- r_smax: záróirányú ellenállás
- atC_d: dióda kapacitása

A ’sav_varikap’ tábla mezői:

- id: elsődleges kulcs
- tipus: dióda típusa, a pontos beazonosításhoz
- bekot: bekötés (melyik irányba vezet)
- labszam: dióda kivezetéseinek száma
- tok: tokozás típusa
- V_Rmax: záróirányú feszültség maximuma
- I_Fmax: nyitóáram maximuma
- C_dmax: dióda kapacitása előírt záróirányú feszültség mellett
- zarfesz: zárófeszültség
- atf_1: minimum frekvencia, ahol az eszköz még üzemszerűen használható
- atf_2: maximum frekvencia, ahol az eszköz még üzemszerűen használható
- r_Dmax: nyitóirányú ellenállás
- V_Fmax: nyitóirányú feszültség maximuma

Az ’alt_dioda’ tábla mezői:

- id: elsődleges kulcs
- tipus: dióda típusa, a pontos beazonosításhoz
- bekot: bekötés (melyik irányba vezet)
- labszam: dióda kivezetéseinek száma
- tok: tokozás típusa
- V_Rmax: záróirányú feszültség maximuma
- I_Fmax: nyitóáram maximuma
- V_Fmax: nyitóirányú feszültség maximum
- nyitoaram: nyitóáram
- t_rr-max: záróirányú éledési idő maximum
- C_dmax: dióda jellemző kapacitása
- zarfeszC_d: jellemző kapacitáshoz tartozó záróirányú feszültség
- I_Rmax: szivárgási egyenáram maximum
- zarfeszI_R: szivárgási egyenáram maximumhoz tartozó záróirányú feszültség

Az ’egyenir’ tábla mezői:

- id: elsődleges kulcs
- tipus: dióda típusa, a pontos beazonosításhoz
- bekot: bekötés (melyik irányba vezet)
- labszam: dióda kivezetéseinek száma
- tok: tokozás típusa
- V_RRM: ismétlődő záróirányú feszültség
- I_FAV: átlagos nyitóáram
- T_rr_max: záróirányú éledési idő maximum
- I_FSM: nem ismétlődő maximális nyitóáram
- V_Fmax: nyitóirányú feszültség maximum
- atI_F: nyitóáram
- I_FRM: ismétlődő maximális nyitóáram
- C_d: dióda kapacitása
- E_RSM: nem ismétlődő lavina energia

7.3 Inkremensek

7.3.1 A keresés

A keresés kettő alapvető csoportra van bontva. Az első eset, amikor ismerjük a diódafajtáját és/vagy típusát, és tudni akarjuk annak tulajdonságait. Ez az egyszerűbb eset, mert nem kell sok keresési feltételt megadni. Egy legördülő listában helyezkednek el a diódák fő csoportjai. Ha tudjuk, hogy melyik osztályba tartozik a dióda, akkor itt ki lehet választani. Ez azért hasznos, mert ezt a lépést követően a keresést csak egy adattáblán kell végrehajtani. Ha ily módon kiválasztottuk a táblát, akkor már csak a megfelelő típust kell kiválasztani a táblából. Egy egyszerű textboxba lehet beírni a típust, kattintani a keresés gombra, és ha a keresés során a program talál egyezést, akkor meg is van, amit kerestünk. Hátránya, hogy megvan a veszélye az elgépelésnek, vagy a típus nevének más formában történő tárolására. Például lehet, hogy a szóköz '_' jellel van helyettesítve, természetesen ekkor nincs egyértelmű találatunk. Egy másik lehetőség, hogy az osztály kiválasztása után egy vagy több másik legördülő listában vannak a típusok felsorolva, ahonnan kiválaszthatjuk a keresett elemet.

A keresés másik nagy csoportja, amikor bizonyos tulajdonságoknak megfelelő diódát kell megtalálni. Ilyenkor ismerni kell az osztályt, hiszen a megfelelő adatokat csak annak ismeretében tudjuk lekérdezni. Ebben az esetben tehát elsőként (az előző esettel ellentétben kötelezően) ki kell választani az osztályt. Ha ez megtörtént, akkor megint marad egy adatbázis, amiben kereshetünk. Itt egy űrlapon kell végigmennünk, és az általunk ismert pontos adatokat, vagy tartományokat kell a keresési feltételekben megadni. Az itt megadott adatok segítségével lesz leszűkítve az adatbázis a számunkra érdekes típusok körére.

Mindkettő keresésnek az eredményét meg kell valahogy jeleníteni. Ez egy külön ablakban nyíló adatlapon lesz, amely akkor jelenik meg, ha rákattintunk a keresés eredményeképpen kiírt típusra. Ez az adatlap már az összes tárolt információt megjeleníti az adott típusról. Ezeket az adatlapokat dinamikusan kell létrehozni különböző függvényekkel.

7.3.2 Regisztráció és bejelentkezés

A weblap menüszerkezete alatt helyeztem el bejelentkezési és regisztrálási funkciót. A bejelentkezés felhasználói névvel és jelszóval történik. Ezek ellenőrzését és a beléptetést login.php fájl valósítja meg. Ha az adott felhasználó szerepel az adatbázisban, akkor a jogosultságának megfelelő oldal töltődik be. A regisztrációs űrlapot egy külön PHP állomány hozza létre egy függvény segítségével. Ugyanez az állomány tartalmazza a regisztráció további folyamataiért felelős kódokat. Ilyenek a megerősítő e-mail küldése, az e-mail cím validálása. Ha a felhasználó nem szerepel az adatbázisban, vagy elgépelt valamit, akkor a megfelelő hibaüzenet jelenik meg.

7.3.3 Jelszócsere, elfelejtett jelszó

Bizonyos rendszerek megkövetelik, hogy rendszeres időközönként a felhasználók megváltoztassák a jelszavaikat. Itt nem kötelező, de a felhasználók módosíthatják a saját jelszavaikat, a modpass.php állomány segítségével. Sokszor az ilyen módosításnak az eredményeképpen, vagy egyszerűen azért, mert régen látogatta a honlapot, előfordul az, hogy a felhasználó elfelejti jelszavát. Ilyenkor lehet kérni újat. A lostpw.php állomány generál új jelszót, és a regisztrációs folyamat utáni első belépés biztonsági szintjének megfelelően segít a felhasználó visszajutásában.

7.3.4 Felhasználók listája (Adminisztrátorok részére)

Az adminisztrációs felületet az admin.php állomány tartalmazza. Bejelentkezés után az adminisztrátor előtt megjelenik az összes tag egy listában. Kiválasztva közülük egyet, módosíthatók, vagy törölhetők az adatok. Az adminisztrátorok beléptetését szintén az admin.php állomány végzi. Ezt a bejelentkezési lehetőséget biztonsági okokból közvetlenül a weblapról nem érhetjük el, nem mutat ide semmilyen link. Csak azok láthatják ezt a felületet, akik tudják, hogy mit kell hozzá a böngésző címsorába beírni, vagy azok, akik véletlenül beírták, de ennek elég kicsi a valószínűsége

7.3.5 Személyes adatok és azok módosítása

A bejelentkezett tagok, ahogy fentebb is említettem, az oldalon több szolgáltatást érnek el, mint az egyszerű látogatók. Ehhez regisztrálni kell, a regisztrációkor pedig adatokat kell megadni. Ha esetleg valamelyik adat változik, akkor a (tagok részére kibővített) menüben érik el az adatmódosító funkciót. A funkció a datamod.php állományban található, melynek segítségével lehet módosítani a felhasználói adatokat. A regisztráció során a felhasználói adatok megadásánál meg lehet jelölni, hogy az aktuális adat nyilvános, vagy privát legyen. Így ha esetleg a későbbiekben a felhasználók meg tudják majd nézni egymás adattábláját, akkor már nem kell a miatt aggódni, hogy valakinek esetleg kellemetlenséget okoztunk, mert az alkalmazás fel lesz készülve ennek a funkciónak egyik legjellemzőbb problémájára.

7.3.6 Regisztráció törlése

Az előbbivel azonos menüből elérhető funkció, szintén ugyanabban a datamod.php állományban került megvalósításra. Bár ez a funkció is a személyes adatok módosításához tartozik, véglegessége miatt külön kell említeni. Használatával (mint a neve is utal rá), minden regisztrált felhasználó saját maga törölheti regisztrációját, ezzel megszüntetve minden vele kapcsolatos adatot a ’user’ táblában. Természetesen itt is szükséges bizonyos biztonsági protokollokat kialakítani, a felhasználónak meg kell erősítenie e-mailben a törlést, hiszen az is történhetett, hogy valaki megszerezte a jelszavát, és kellemetlenséget akar okozni.

7.3.7 Üzenőfal

A tagoknak biztosítani kell egy lehetőséget az alkalmazáson belüli csoportos kommunikációra, az nem megoldás, hogy mindenki tegye nyilvánossá az e-mail címét. Ilyen jellegű oldalak felhasználói között komoly tudásbeli különbség lehet. Ha van egy ilyen kommunikációs lehetőség, akkor egymástól lehet kérdezni, egymáson lehet segíteni, ezzel esetleg olyanokat is kisegítve, akik nem merik feltenni kérdéseiket. Ha a lapnak esetleg sikere lesz és kicsit kinövi magát, akkor növelni kell az adminisztrátorok számát. Ilyen esetekre is jó az üzenőfal tartalmát áttekinteni, mert sok esetben a segítségnyújtásban legaktívabb fórumozókból lesznek a későbbi legjobb adminisztrátorok, ha vállalják a felkérést.

7.4 Inkremensek validálása

Ha elkészült egy inkremens, akkor a következő lépésben azt validálni kell. Itt kerül ellenőrzésre, hogy az inkremens által implementált funkciók megfelelően kerültek-e kivitelezésre, és tényleg a specifikációban rögzített szempontok az optimálisak a probléma szempontjából. Van úgy, hogy a specifikáció pontosításra szorul, mert kiderül, hogy más megoldással hatékonyabb rendszert kapnánk. Az is előfordulhat, hogy a megvalósítás szorul javításra.

7.5 Rendszervalidálás

A rendszerinkremensek megvalósítása után meg kell vizsgálni, hogy lehetne-e úgy bővíteni a rendszert, hogy még jobban elláthassa feladatát. Meg kell vizsgálni, hogy a kész rendszer eleget tesz a megfogalmazott funkcionális és nem funkcionális követelményeknek. Ezt akkor jelenthetjük ki biztosan, ha a rendszer éles környezetben sikeresen be van üzemelve és üzemzavarmentesen használható. Persze további bővítésekre még ilyenkor is lehet javaslatot tenni.

8. A rendszer bemutatása

A felhasználók két csoportját különböztetjük meg.
Ezek:

- Látogatók
- Tagok

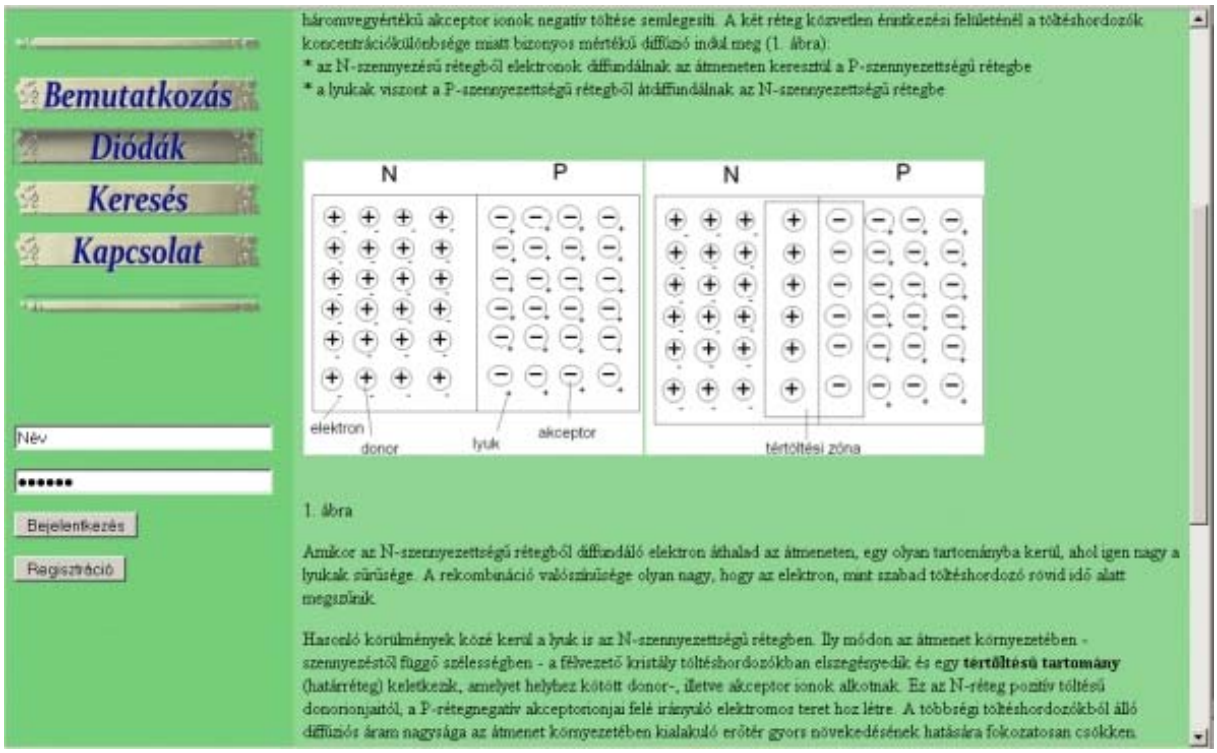
A két csoport eltérő jogokkal rendelkezik. A látogatóknak nincs más dolguk, mint regisztráció nélkül böngészni az oldalak között és használni az adatbázist. A ’Kapcsolat’ menüpontot viszont nem tudják használni. Erre azért van szükség, mert egy nem ellenőrzött üzenőfalra sajnos olyan tartalmak kerülhetnek, amelyek az oldal szerkesztőjét érintenék kellemetlenül. A tagokkal kicsit később foglalkozunk, a regisztrációnál.

8.1 Kezdőlap, alapvető funkciók

A kezdőlapon kettő *frame* található. A bal oldali *frame* nem változik, ott a menü található, a navigációnak így fix helye van. A jobb oldalon egy bemutatkozó *frame* nyílik meg. Egy kis leírást tartalmaz a weblapról, itt lesz a felhasználói dokumentáció a könnyebb kezelés érdekében, és később ez lesz a hírek helye. Hírek alatt a fejlesztéssel kapcsolatos előrelépéseket, az adatbázis említésre méltó bővülését, esetleg a fenntartással kapcsolatos információkat értem.

15. ábra – Kezdőlap

A menüpontokra kattintva a jobb oldali *frame* tartalmaz minden megjelenő lapot. A következő menüpont, a ’Diódák’ szintén mindenki számára elérhető. Itt egy leírás található a félvezetőkről, ami az alapoktól kezdi a témakört, hogy azok is megértsék a működési elvüket, akik eddig nem tanultak elektronikát. Több oldalon keresztül fogja tárgyalni a félvezetőkről szóló fejezeteket, hogy mindenki számára egyértelmű lehessen a téma.



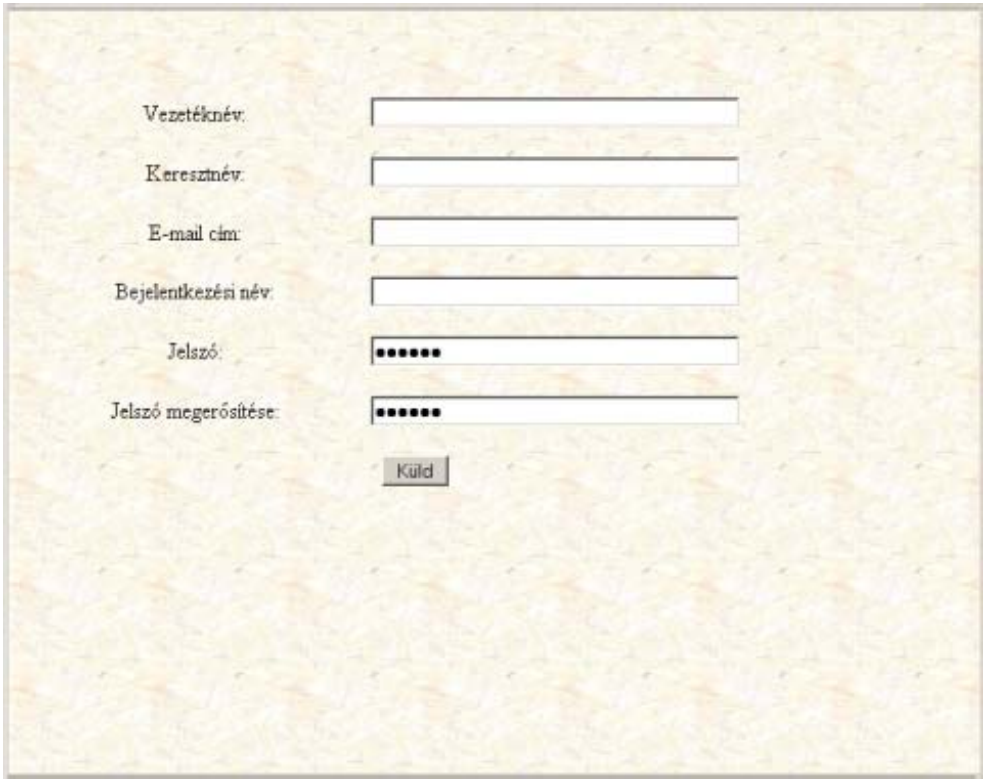
16. ábra - Diódák-lap

A keresés funkció az adatbázis lekérdezéseihez vezet. Ez a funkció szintén elérhető bárki számára, nem szükséges a regisztráció. Első körben a típus szerinti keresés valósul meg. Az osztály kiválasztásával kezdjük, ezzel máris kiválasztva az aktuális táblát. Ha nem tudjuk, milyen osztályba tartozik a keresett dióda, akkor minden táblát át kell vizsgálni. Az osztály kiválasztása egy legördülő menüvel van megvalósítva. Azzal lehet kiválasztani az osztálynak megfelelő adattáblát. Utána a legördülő menü alatt látható egy *inputbox*, mellette egy 'Keres' feliratú gombbal. Ide is beírhatjuk a típust, amit keresünk, de megvan az esélye, hogy elírjuk, így azt kapjuk majd vissza, hogy az adatbázisban nem található ilyen értékű elem. Ameddig az adatbázis csak kevesebb adattal van feltöltve, addig a gomb alatt megjelenő összes találat átnézése sem okozhat gondot, sőt addig még biztosabb is. Ha már sok adat lesz, akkor nehezebb lesz áttekinteni a teljes találati listát, akkor egy kis módosítással be lehet építeni a rendszerbe egy szűrést, ami megkönnyítheti a helyzetünket.

17. ábra - A keresés

8.2 Regisztráció és bejelentkezés

A regisztráció művelete alakítja át a látogatót taggá. Itt lehet megadni egy bejelentkezési azonosítót, amely a későbbiekben úgynevezett *nicknév* szerepét tölti be az azonosításban. A Menük alatt található a ’Regisztráció’ gomb, ami feldob egy ablakot, ami tartalmaz egy űrlapot (**18. ábra**). Az űrlap kitöltése közben felmerülhet, hogy a bejelentkezési azonosító már foglalt. Ilyenkor a rendszer küld egy figyelmeztetést erre vonatkozóan és addig nem engedi folytatni a regisztrációt, ameddig nem helyettesítjük valami mással. A többi adatban lehet egyezés, kivéve az e-mail címet, de elvileg ez mindenkinek egyedi. Ezt azért érdemes vizsgálni, mert ha kettő regisztráció szerepel ugyanazzal az e-mail címmel, az nem egészséges, leginkább a felhasználók azonosítása miatt. Egy felhasználó egy regisztrációval rendelkezhet, nem is érdemes többel, mert semmi haszna nem lehet belőle. Ha mégis többre vágyik, akkor azt úgyis megoldja több e-mail fiókkal. Ráadásul az e-mail cím fontos szerepet tölt be az azonosításban és az elveszett jelszó módosításában, ezért a felhasználókat tartalmazó adattáblában biztos, hogy nem szerepelhet több azonos e-mail cím.



18. ábra - Regisztrációs űrlap

A regisztrációs űrlap sikeres kitöltését követően a leendő tag kap egy e-mailt a megerősítő kóddal, ami egy legenerált véletlen szám, csak a rendszer tudja, és aki megkapta. Az első bejelentkezéskor van szükség erre a számra, azután törlődik a rendszerből.

A regisztrációs folyamatnak ezzel a végére is értünk, ettől a ponttól már be lehet jelentkezni. A bejelentkezéshez szintén a menüfelület biztosítja a lehetőséget. Ott vár minket kettő inputbox, alattuk egy ’Bejelentkezés’ gombbal. Sikertelen bejelentkezési kísérlet esetén a rendszer felajánlja a javítási lehetőséget, és az elfelejtett jelszó helyett új küldésének lehetőségét e-mailen keresztül. Bejelentkezés után első látásra minimális változás történik. Mivel semmi szükség nincs rá, eltűnik a ’Regisztráció’ gomb, helyette megjelenik a ’Saját adatok módosítása’ link, ami egy hasonló ablakot jelenít meg, mint a regisztrációs űrlap, csak kevesebb adatkéréssel. Módosítani az e-mail címet és a jelszót lehet, és bekerül egy új funkció, a regisztráció törlése. Ez is csak megerősítő e-mail közreműködésével engedélyezett funkció. Természetesen a bejelentkezési lehetőséget a kijelentkezési lehetőség váltja fel a menü alatt.

8.3 Üzenőfal

A tagoknak van egy lehetőségük a nyilvános kommunikációra az üzenőfal segítségével. Itt láthatjuk az eddigi üzeneteket és írhatunk újakat. Ha széles körben elterjedne ennek használata, akkor természetesen érdemes lenne ennek kiváltására egy továbbfejlesztett, strukturált fórumrendszert létrehozni, ahol témák között is lehet válogatni.

8.4 Összegzés

Az mindenképpen bebizonyosodott, hogy nem lehetetlen az, hogy megszülessen egy webes diódakatalógus, amely mögött többtáblás adatbázisban vannak eltárolva a legfontosabb műszaki jellemzők. Lehetséges keresni típus alapján konkrét diódára, hogy megnézhessük az adattábláját, vagy lehet keresni tulajdonságok alapján, hogy megtudjuk, milyen típusra is van szükségünk. Az üzenőfal talán legfontosabb szerepe, hogy közvetíteni tudja a tagok gondolatait, ötleteit a további fejlesztésekhez.

9. Befejezés

Szakdolgozatom készítése közben nagyon sokat tanultam a webes alkalmazásfejlesztés különböző szakaszairól, a tervezés és a megvalósítás során alkalmazott technológiákról, módszerekről, modellekről. A dolgozat végén már egy webes diódakatalógus alapjait próbáltam lefektetni.

A szakdolgozatom végül elérte az általam kitűzött célokat. Az alkalmazott technológiák és az alapfogalmak gyors áttekintése után szépen lépésenként a modellezéstől elkezdve a szoftvertervezés és fejlesztés folyamatát már egészen részletesen sikerült bemutatni. A létrehozott alkalmazás meglévő funkciói működnek, de még további fejlesztéseket, bővítéseket igényel. Éles környezetben még nem történt meg a tesztelése, későbbiekben tervezem beüzemelni, ha sikerült egy dekoratívabb stílust kialakítani rajta. Az alkalmazás bővítése a szerkezetének köszönhetően viszonylag könnyen lehetséges, akár komplett modulokkal is.

A témaválasztásom nagy kihívás elé állított, hiszen egy összetett és bonyolult feladatot adtam ezzel magamnak. Rengeteg probléma állt az utamba, melyeket meg kellett oldani, vagy meg kellett kerülni. Ilyenek voltak a böngészők közötti különbségek a kód értelmezését tekintve, vagy a regisztrációs folyamat részletei. Ami igazán izgalmas, megtanulni megoldani azokat a problémákat is, amelyek első látásra csak megkerülhetők.

10. Irodalomjegyzék

1. „Rövid betekintés a Html 3.2-be és a JavaScriptbe” - Dvorák Péter és Hőrcsik Loránd
   http://nyf.beckground.hu/incoming/klte-jegyzetek/Jegyzetek_Konyvek/HTML/html.doc
   2009.04.22

2. „SSADM strukturált rendszerelmezési és tervezési módszer” – Kincses László
   http://www.itb.hu/ajanlasok/a4/
   2009. 04. 20.

3. „Szoftvertechnológia” - Szabolcsi Judit
   http://www.gamf.hu/portal/files/szft_1_2.pdf
   2009.04.25.

4. Nagy Gusztáv – Web programozás
   http://nagygusztav.hu/sites/default/files/Web_programozas_jegyzet_0.7_0.pdf
   2009. 04. 07.

5. Kondorosi Károly, Szirmay-Kalos László, László Zoltán: Objektum orientált szoftverfejlesztés
   http://www.tankonyvtar.hu/main.php?objectID=5331783
   2009. 04. 19.

6. PROG.HU - Magyar karakterek a weben és PHP-ben
   http://prog.hu/cikkek/873/Magyar+karakterek+a+weben+es+PHP-ben.html
   2009. 04. 08.

7. PHP online dokumentáció
   http://www.php.net/manual/hu/
   2009. 04. 20.