13. Tétel

a) Webprogramozás II.

A dinamikus weboldalak jellemzői, összehasonlításuk a statikus tartalmakkal

- Dinamikus:

- adatkinyerés és manipuláció

- interaktív funkciók

- felhasználói funkciók

- Jellemzői: rugalmas, interaktív, nagyobb komplexitású és lassabb betöltési idő

- Megvalósítás: PHP, Node.js szerveroldali technológiák

- Statikus:

- előre gyártott tartalom

- gyors betöltés

- könnyű karbantarthatóság

- Jellemzői: gyors, korlátozott

A PHP nyelv jellemzői, használatának feltételei a webfejlesztés során

- szerveroldali szkriptnyelv

- keresztplatformos Apache, Nginx, Windows, Linux

- beágyazható HTML kódba

- nyílt forráskódú

- adatbázis támogatás: MySQL, PostgreSQL, Oracle stb.

- fejlesztői környezet szükséges (Visual Studio Code)

- <?php ?>, $ változó, integer, float, string, boolean, vezérlési szerkezetek ugyanazok, pl. HTTP kérések feldolgozására, keretrendszerek/szabványosítás (pl. Laravel)

Kliens és szerver oldali infrastruktúra a PHP futtatásához

- Szerveroldali infrastruktúra:

- Szükséges egy webserver ami képes értelmezni és végrehajtani a PHP kódot: Apache, Nginx

- Szükséges egy adatbázis-kezelő rendszer: MySQL, PostgreSQL, SQLite stb.

- Kliensoldali infrastruktúra:

- Webböngésző: felhasználók ezen keresztül kommunikálnak a szerverrel és ezek jelenítik meg a dinamikus weboldalakat

- JavaScript: felhasználói felületek interaktivitásához

- Oldalak/Nézetek (HTML) és Stíluslapok (CSS)

A PHP típusrendszere: típusok, konstansok, változók a PHP-ban

- Típusok:

- int, float, string, bool, array, object, null stb. támogatja

- erős és gyenge típusellenőrzés azaz automatikusan meghatározódnak az érték alapján

- Konstansok:

- szokásos dolgok

- define() kulcsszó, const segítségével

- névkonvenció: UPPER\_CASE\_WITH\_UNSERSCORES

- Változók:

- szokásos dolgok

- névkonvenció: camelCase vagy snake\_case

- globális és lokális változók

A HTML és a PHP kapcsolata

- PHP: oldal feldolgozásáért és generálásáért

- HTML: tartalom megjelenítéséért

- a PHP beágyazható a HTML kódba

- Példa rá: Form/Űrlap-> a HTML űrlapon adatokat küldünk a szerver felé, a PHP pedig feldolgozza azokat

- űrlap fejléce: <form action="procces.php" method="post" ... </form>

- CRUD műveletek támogatása

- Session és Cookies használata, állapotátadás, munkamenetkezelésre

Modulszerkezet kialakítása, a forráskód újrahasznosításának lehetőségei

- külön funkcionális modulok pl. felhasználókezelés és a termékekhez

- komponensek szétválasztása pl. header, footer

Újrahasznosítások lehetőségei:

- include, require

- Osztályok és objektumok használata azaz OOP alapelvei (3)

Adatcsere a kliens és szerver oldal között

- kulcsfontosságú a dinamikus weboldalaknál

- pl. űrlapok segítségével / CRUD műveletek

- Ajax kérések: aszinkron adatcsere kliens és szerver oldal között, az oldal újratöltése nélkül

- munkamenetek sütik kezelése (session\_start() $\_SESSION["username"] = "john";)

- Süti: kis adatdarabok amelyeket a szerver a böngészőbe küld, a böngésző pedig visszaküldi a kérések során. (kliens oldali adattárolás)

- JSON és XML: olyan adatcsere szabványok, amelyek az objektumokat tömbként ábrázolja

- WebSockets használata: valósidejű kétirányú kommunikáció

Biztonsági kérdések

- Adatvédelem:

- adatvédelmi szabályozások betartása (GDRP)

- érzékeny adatok tárolása (titkosítás)

- Felhasználói Hitelesítés és Jogosultságkezelés

- erős jelszavak, kettős hitelesítés, CATCPHA

- csak azokhoz az adatokhoz és funkciókhoz férjünk hozzá amire jogosultak vagyunk

- XSS és CSRF támadások elleni védelem

- Biztonságos Kommunikáció (SSL/TLS titkosítás kliens és szerver között) valamint HTTPS protokoll

- Hibakezelés és logolás

Munkafolyamatok és sütik kezelése

- fentebb volt róla szó

- Munkamenetek: adatok tárolása a szerveroldalon a felhasználó böngészési ülése alatt

- Sütik: adatok tárolása felhasználó kliens oldalán

b) Programozási technológiák

Tervezési minták szerepe és osztályozása a szoftverfejlesztésben

- a tervezési minták gyakori programozói feladatokat oldanak meg

- GOF könyvben

- könnyen bővíthető, módosítható osztályszerkezetet kapunk, tehát rugalmas kódot

- másik megközelítése az extrém programozás

- a tervezési minták ára a bonyolultabb kód, mert olyan osztályok bevezetésére is kell gondolnunk amiknek semmi köze a valósághoz. (ez kicsit ellentétes az OOP-vel, mert az OOP osztály a valóság absztrakciója)

A stratégia és a sablon metódus tervezési minta összehasonlítása:

- Sablon:

- ekkor használjuk: van egy átalános recept->több hasonló dolog gyártható (pl. tea és a kávé)

- receptben háromféle lépés:

- kötelező és közös: ez a lépés mindig ugyanaz (ősben definiáltak, pl. Vízforraló metódus)

- kötelező, de nem közös: szükséges, de minden esetben más és mást kell csinálni (ősben absztrakt metódusok, pl. édesítés)

- opcionális: nem minden esetben szükséges (ősben hook metódusok, azaz van törzsük, de üres pl. virtuális metódusok)

- maga a recept a sablonfüggvény, védelmi szintek alkalmazása a metódusokon, a sorrend tartása miatt) pl. rum();

- IoC, az absztrakt ős és a gyermek osztály között, nem a gyermek hívja az ős metódusait hanem az ős a gyermekét (ez amiatt van mert a sablonfüggvény absztrakt és virtuális metódusokat hív)

- Stratégia:

- algoritmus családok definiálása és innen választunk a módszereknél

- viselkedési formáink vannak és ezekkel ruházzuk fel

- Összehasonlítás:

- Stratégia: ugyanazt csináljuk, de másképp

- Sablon: ugyanúgy csináljuk, de mást

a megfigyelő tervezési minta szerepe és fajtái, egyéb tervezési minták

- Megfigyelő: objektum megváltozása -> értesít más objektumot

- alany: tárolja a regisztrált megfigyelőket, interfész kínál a ki és beregisztrálásra

- megfigyelő: interfészt definiál azoknak az objektumok számára akik értesülni szeretnénk az alanyban bekövetkezett változásról (update metódus szolgál erre)

- Két fajta megfigyelő megvalósítás:

- Pull-os: a megfigyelő lehúzza a változásokat az alanytól

- Push-os: az alany odanyomja a változásokat a megfigyelőnek

Az objektumorientált tervezési alapelvek szerepe a szoftverfejlesztésben

- a rendszertervezés alapelve: There is no Silver Bullet (a módszertanokról szól és hogy kombinálva jobbak)

- modulok/osztályok megjelenése

- hasonlít a rekord adattípushoz mert inhomogén, de az osztály tartalmazhat eljárásokat függvényeket, a rekord nem

- EZ IDÁIG CSAK ÁLTALÁNOS

- az oop tervezés alapelvei a tervezési mintánál magasabb absztrakciós szinten írja le, hogy milyen a jó program.

- arra jó hogy kiválasszuk az oop alapelvei (öröklődés...) közül azt az eszközt ami a legjobb kódot eredményezi

- cél: bővíthető, újrafelhasználható könnyen érthető más programozók számára is

a nyitva-zárt alapelv bemutatása

- OCP: a program forráskódja nyitott a bővítésre, de zárt a módosításra

- lehet érteni az osztályhierarchiára is (ilyenkor még szigorúbban a szintaxis szintjén is meglehet fogalmazni)

- pl. C#: ne használd az override kulcsszót, kivéve ha absztrakt vagy hook metódust akarsz felülírni

- azért nem szabad megszegni mert ha van egy jól tesztelt működő jól kiforrott metódusod, azt megváltoztatva több negatív dolog is lehet

- jó példa rá a stratégia és a sablon (hookot is használunk)

a GOF1 és a GOF2 alapelvek rövid bemutatása, egyéb alapelvek

- GOF1: 1995 Gang of Four: ez egy fontos alapelv: "Programozz felületre implementáció helyett"

- akkor programozunk implementációra ha kihasználjuk hogy egy osztály hogyan lett implementálva

- Tehát, ha a kódunkban találunk olyan részt, amely egy másik osztály implementációjától függ, akkor az hibás tervre utal

- Ha implementációra programozunk, és ha megváltozik az osztály, akkor a vele kapcsolatban álló osztályoknak is változniuk kell. Ezzel szemben, ha felületre programozunk, és megváltozik az implementáció, de a felület nem, akkor nem kell megváltoztatni a többi osztályt.

- GOF2: fontos alapelv: "Használj objektum összetételt öröklés helyett, ha csak lehet"

- öröklődés: IS-A kapcsolat (Kutya, Gerinces) -> az osztály összes szolgáltatását megörökli -> átlátszó újrahasznosítás

- objektum összetétel: HAS-A kapcsolat (Kutya, Gerinc) -> csak az osztály egy példányára szerzek referenciát és azon keresztül használjuk a szolgáltatásait -> átlátszatlan újrahasznosítás

- több fajtája van: Aggregáció(Gitáros dead->gitár->temetik), Kompozíció(Gitáros dead->gitár->eldobják), Becsomagolás (átlátszó-> A karácsonyfa karácsonyfa

marad, akárhány díszt is teszek rá és átlátszatlan->a becsomagolt karácsonyfa nem marad karácsonyfa)

- OCP, LSP(a program viselkedése nem változik meg attól, hogy az ős osztály helyett egyik gyermek osztályának példányát használom), DIP, HP, Demeter törvénye / a legkisebb tudás elve

A jól bevált módszerek szerepe a szoftverfejlesztésben

- RFT-ből ismert módszerek

a TDD bemutatása, egyéb jól bevált módszerek

- A TDD egy iteratív szoftverfejlesztési módszertan, amelyben egy új funkció implementálása előtt írjuk meg az ahhoz tartozó automatizált teszteket. Ez egy ciklikus folyamat, amely az alább ismertetett lépésekből áll. A tesztek íródnak meg először, utána pedig a kódok amik kielégítik a tesztet. A kódokat és teszteket ezután kijavítjuk és így megy addig amíg számunkra megfelelő eredményt nem kapunk

- lépései: teszt hozzáadása, minden korábbi teszt futása, kód írás, minden teszt újbóli futtatása, kódszépítés

- egyéb módszerek: ATDD (Acceptance...) Elfogadási ...

- Az ATDD egy teszt-első megközelítés. A teszteseteket a felhasználói történet megvalósítása előtt készítik el. A teszteseteket a csapat különböző nézőpontú tagjai, például az ügyfelek, a fejlesztők és a tesztelők készítik. A teszteseteket kézzel vagy automatizáltan lehet végrehajtani.

- A BDD egy szoftverfejlesztési folyamat, amely az agilis módszertanon alapul. A koncepciója a TDD megközelítéséből származik. Ez a szoftverfejlesztési folyamat a végfelhasználói követelményekre és a termékkel való interakciókra összpontosít. Mennyire jól fognak működni az alkalmazás alapvető funkciói a végfelhasználók számára, milyen lesz az alkalmazás általános viselkedése a felhasználók számára, és milyenek a közönség visszajelzései az alkalmazással való interakció során - mindezeket a tényezőket figyelembe vesszük a BDD megközelítéssel történő termékfejlesztés során.

YAPPING:

Mint már írtuk, a TDD másik neve a Piros – Zöld – Piros tesztelés. A fenti esettanulmányban nagyon jól látható, hogy a Szum metódust lépésről lépésre írjuk meg, úgy, hogy először megírjuk az új esetet lefedő egységtesztet, majd az ezt megvalósító kódot.

A fenti esettanulmányból nem látszik viszont, hogy néha szépítjük is (idegen szóval refaktoráljuk) a kódot. Ha azt vesszük észre, hogy a kódban redundáns sorok vannak, vagy lehetne alkalmazni egy tervezési mintát, esetleg szét lehetne választani valamit, amire eddig nem volt szükség, akkor kódszépítés következik. Ez a kék fázis. Elvileg minden zöld lépés után kék jön, ezért a TDD-t nevezhetjük Piros – Zöld – Kék – Piros tesztelésnek is.