Dr. Mileff Péter

Szoftverfejlesztés

Prototípus készítés, tervezés

1

Szoftverprototípus készítése

- **A prototípus fogalma:**
  - **a szoftverrendszer kezdeti verziója,**
  - **Arra használják, hogy bemutassák a koncepciókat,**
  - **kipróbálják a tervezési opciókat,**
  - **és hogy jobban megismerjék a problémát és annak lehetséges megoldásait.**
  - A prototípus gyors, iteratív fejlesztése azért nagyon fontos, mert a költségek így ellenőrizhetők.

2

Szoftverprototípus készítése

- A szoftverprototípus több helyen is használható a fejlesztés folyamatában:
  - 1. A követelménytervezés folyamatában.
  - 2. A rendszertervezési folyamatban.
  - 3. A tesztelési folyamatban.
- A követelmény fázisban:
  - a prototípus fejlesztése alatt fény derülhet a követelményekkel kapcsolatos lehetséges hibákra,
  - esetleg új ötletek kapcsán új rendszerkövetelményeket is indítványozhatnak.

3

Szoftverprototípus készítése

- A rendszertervezési folyamatban:
  - a rendszerprototípus felhasználható a tervezési tapasztalatok alkalmazására,
  - illetve a javasolt terv megvalósíthatóságának felülvizsgálatára.
  - Pl.:
    - a gyors prototípus készítés az egyetlen módja annak, hogy a felhasználók bevonásával grafikus felhasználói felületeket fejlesszünk ki.

4

Szoftverprototípus készítése

- A rendszertesztelés fázisában:
  - a prototípusok segítségével az eredmények ellenőrzéséhez szükséges munka *visszacsatoló tesztek* segítségével csökkenthető.
  - Ilyenkor ugyanazokra a tesztesetekre futtatjuk a prototípust és a rendszert.
    - Ha mindkét rendszer ugyanazt az eredményt adja, akkor a teszteset valószínűleg nem talált hibát.

5

Szoftverprototípus készítése

- A prototípuskészítés rendszerint a szoftverfolyamat korai szakaszában növeli a költségeket,
  - később viszont jelentősen csökkenti, mert elkerülhetők az újraírási munkák.
- A prototípuskészítés folyamata **négy** különböző fázissal írható le:
  - A prototípus céljainak megállapítása
  - A prototípus funkcionalitásának definiálása
  - A prototípus fejlesztése
  - A prototípus kiértékelése

6

A négy fázis

- 1. A prototípuskészítés céljait érdemes az elején írásban megadni,
  - e nélkül a vezetés vagy a végfelhasználók félreérthetik a rendeltetését.
  - Ilyen cél lehet:
    - az alkalmazás megvalósíthatóságának demonstrálása vagy a felhasználói felületek bemutatása, stb.
  - Ugyanaz a prototípus nem szolgálhatja az összes célt.
- 2. Annak eldöntése, hogy mit tegyünk bele a prototípusba.
- 3. Prototípus fejlesztése
- 4. A kiértékelés
  - Gondoskodni kell a felhasználók képzéséről,
  - mivel időbe telik, amíg megszokják az új rendszert, és csak ezután fedezhetik fel a követelménybeli hibákat és hiányosságokat.

7

A két fő típus

- Két fő típusa: **evolúciós** és az **eldobható**.
- Az **evolúciós prototípus** készítésének célja:
  - egy működő rendszer átadása a végfelhasználóknak.
  - Ezért a legjobban megértett és leginkább előtérbe helyezett követelményekkel javallott kezdeni.
  - Kevésbé fontos és körvonalazatlanabb követelmények:
    - akkor kerülnek megvalósításra, amikor a felhasználók kérik.
  - Ez a módszer a weblapfejlesztés és az e-kereskedelmi alkalmazások szokásos technikája.

8

A két fő típus

- Az **eldobható prototípus** készítésének célja:
  - a rendszerkövetelmények validálása vagy származtatása.
  - A nem jól megértett követelményekkel érdemes kezdeni, mivel azokról szeretnénk többet megtudni.

9

TERVEZÉS…

10

Bevezetés

- **A szoftvertervezés lényege a szoftver logikai szerkezetére vonatkozó döntések meghozatala.**
- Bizonyos esetekben ezt a logikai szerkezetet egy olyan modellezőnyelv segítségével leírt modellel ábrázoljuk, mint amilyen az UML.
- Más esetekben a tervet informális jelölésrendszerrel leírt vázlatokkal reprezentáljuk.

11

ARCHITEKTÚRÁLIS TERVEZÉS…

12

Architektúrális tervezés

- A nagy rendszereket olyan alrendszerekre bontják, amelyek biztosítják az egymással kapcsolatban lévő szolgáltatásokat.
- Ezen alrendszerek azonosításának és az alrendszer vezérlésére és kommunikációjára szolgáló keretrendszer létrehozásának kezdeti tervezési folyamata az **architektúrális tervezés**.

13

Architektúrális tervezés

- Egy kreatív folyamat, ahol megpróbálunk előállítani egy rendszerszerkezetet:
  - amely megfelel a funkcionális és nemfunkcionális rendszerkövetelményeknek.
- A rendszer architektúrája befolyásolja
  - a rendszer teljesítményét,
  - robusztusságát,
  - elosztható ságát és
  - karbantarthatóságát.
- Ezért fontos:
  - a szoftvertervezőket rákényszerítsük arra, hogy a tervezés kulcsfontosságú aspektusaival már a folyamat korai szakaszában foglalkozzanak.

14

Architektúrális tervezés

- Az alkalmazás számára választott szerkezet:
  - Függhet nemfunkcionális rendszerkövetelményektől is
  - Pl.: a *teljesítmény*, *védettség*, *biztonságosság*, *rendelkezésre állás*, és *karbantarthatóság*.
- Magában foglalja a rendszerek alrendszerekre bontását is
  - Az alrendszerek tervezése tulajdonképpen a rendszer durva szemcsézettségű komponensekre történő absztrakt felbontása,
  - Ezen komponensek lehetnek önálló rendszerek.
  - Az alrendszerek terveit általában blokkdiagram segítségével írjuk le, ahol a diagram dobozai az egyes alrendszereket reprezentálják.

15

Tervezési döntések

- A tervezési folyamat során a rendszer tervezőinek számos döntést kell meghozniuk.
  - amelyek alapvetően kihatnak a rendszerre és a fejlesztési folyamatra.
- Tudásuk és tapasztalataik alapján a következő alapvető kérdésekre kell válaszolniuk:
  - 1. Létezik-e olyan általános alkalmazás architektúra, amely a tervezendő rendszer számára mintául szolgálhat?
  - 2. Hogyan osztjuk szét a rendszert processzorokra?
  - 3. Milyen architektúrális stílus lenne a rendszer számára megfelelő?

16

Tervezési döntések

- 4. Milyen alapvető megoldásokat alkalmazunk a rendszer strukturálására?
- 5. Hogyan lehet a rendszer szerkezeti egységeit modulokra felbontani?
- 6. Milyen stratégiát kell alkalmazni az egységek működésének vezérlésével kapcsolatban?
- 7. Hogyan értékelik majd ki az architektúrális tervet?
- 8. Hogyan kell a rendszer-architektúrát dokumentálni?

17

Tervezési döntések eredménye

- A folyamat eredménye:
  - egy **architektúrális tervezési dokumentum**,
    - tartalmazza a rendszer grafikus reprezentációit és a hozzájuk kapcsolódó leíró szöveget.
    - Tartalmaznia kell annak leírását, hogy a rendszert hogyan lehet alrendszerekre bontani,
      - az egyes alrendszerek miképpen oszthatók modulokra.

18

Architektúrális modellek

- 1. ***Statikus szerkezeti modell:*** megmutatja, hogyan lehet az alrendszereket és komponenseket különálló egységként fejleszteni.
- 2. ***Dinamikus folyamatmodell:*** ábrázolja, hogy a rendszer hogyan szervezhető futási idejű folyamatokba. Ez különbözhet a statikus modelltől.
- 3. ***Interfészmodell:*** az alrendszerek által publikus interfészeiken keresztül nyújtott szolgáltatásait írja le.
- 4. ***Kapcsolatmodellek:*** az alrendszerek közötti kapcsolatokat (például adatáramlás) mutatják be.
- 5. ***Elosztási modell:*** azt adja meg, hogy az egyes alrendszereket hogyan kell elosztani a számítógépek között.
- A rendszer-architektúrák leírása:
  - számos kutató az architektúraleíró nyelvek (architectural description languages, ADL) használatát és az UML nyelvet javasolja.

19

A rendszer felépítése

- Már az architekturális tervezési folyamat korai szakaszában döntenünk kell a rendszer teljes szerkezeti modelljéről.
- Ezt a szervezési módot az alrendszerek közvetlenül is tükrözhetik,
  - de gyakran előfordul, hogy az alrendszer modellje részletesebb a szerkezeti modellnél.
- A rendszer felépítését számos modell segítségével írhatjuk le.

20

A tárolási modell

- A rendszert felépítő alrendszereknek a hatékony együttműködés céljából információt kell cserélniük
- Két módon történhet:
  - 1. Minden megosztott adatot a minden alrendszer által elérhető, központi adatbázisban kell elhelyezni.
    - **A megosztott adatbázison alapuló rendszermodellt *tárolási modellnek* nevezzük.**
  - 2. Minden alrendszer saját adatbázist tart fent.
    - Ekkor az egyéb alrendszerek közötti adatcsere üzenetküldés segítségével történik.

21

A tárolási modell

- A nagy mennyiségű adatokkal dolgozó rendszerek osztott adatbázisok és tárolók köré szerveződnek.
- Ez a modell azon alkalmazások számára megfelelő,
  - ahol az egyik alrendszerben keletkező adatok egy másik alrendszerben kerülnek felhasználásra.

22

Előnyök hátrányok

- 1. Hatékony módszer nagy mennyiségű adat megosztására.
  - nem szükséges az adatokat explicit módon átvinni egyik alrendszerből a másikba.
- 2. Az alrendszereknek azonban azonos adattárolási modellel kell rendelkezniük.
  - Ekkor kompromisszumokra van szükség az egyes eszközök között,
  - Ez azonban rossz irányban befolyásolhatja a teljesítményt.
  - A közös sémának nem megfelelő adatmodellel rendelkező új alrendszerek integrálása nehézzé vagy lehetetlenné válik.

23

Előnyök hátrányok

- 3. Az adatokat termelő alrendszereknek foglalkozniuk kell azzal, hogy a többi alrendszer hogyan fogja használni azokat az adatokat.
- 4. Az olyan tevékenységek, mint a biztonsági mentés, a védelem, a hozzáférés-szabályozás és a hiba utáni visszaállítás központosítottak,
  - így az eszközök lényeges feladataikra összpontosíthatnak.

24

Előnyök hátrányok

- 5. A tárolási modell viszont ugyanazt a politikát kényszeríti rá minden alrendszerre.
- 6. A megosztottság modellje a tárolási sémán keresztül látható.
  - Ez egyenes utat biztosít az új eszközök integrálásához, amennyiben azok megfelelnek a közös adatmodellnek.
- 7. A tároló több gép közötti elosztása azonban bonyolult is lehet.
  - Bár lehetőség van egy logikailag központosított tároló elosztására, problémák léphetnek fel az adatok redundanciájával és inkonzisztenciájával kapcsolatban.

25

A KLIENS-SZERVER MODELL…

26

Kliens-szerver modell

- A kliens-szerver architektúra modellje egy osztott rendszer modellje.
- Megmutatja, hogy az adat és a feldolgozás hogyan oszlik meg feldolgozóegységek között.
- A modell három fő komponensből, a kliensek és szerverek halmazából, valamint a hálózatból tevődik össze.

27

Kliens-szerver modell szereplői

- **Szerverek halmaza:** amelyek más alrendszerek számára szolgáltatásokat nyújtanak.
  - Pl.: a nyomtatószerverek.
- **Kliensek halmaza:** amelyek hozzáférnek a szerverek által biztosított szolgáltatásokhoz.
  - Ezek általában önálló létjogosultsággal rendelkező alrendszerek. Egy kliensprogramnak számos példánya futhat egyidejűleg.
- **Hálózat:** lehetővé teszi, hogy a kliensek hozzáférjenek a szolgáltatásokhoz.
  - Ez nem szükséges akkor, ha mind a kliensek, mind pedig a szerverek egyetlen gépen futnak.

28

Kliens-szerver modell

- A klienseknek szükségük van arra:
  - ismerjék az elérhető szerverek.
  - az általuk biztosított szolgáltatások neveit
  - de a szervereknek nem kell tudniuk sem a kliens azonosságát, sem pedig azt, hogy hány kliens van.
- A kliensek a szerverek által biztosított szolgáltatásokat távoli eljáráshívásokkal érik el
  - egy kérés-válasz alapú protokoll segítségével,
  - mint pl. a www esetén használt http protokoll.
- Lényegében a kliens elküld egy kérést a szervernek, és addig vár, amíg választ nem kap.

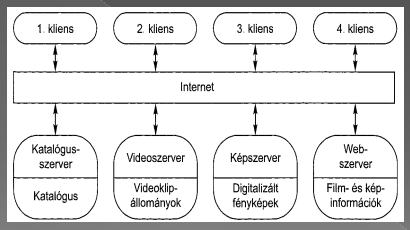
29

Kliens-szerver modell

- A kliens-szerver modell legfontosabb előnye az osztott architektúrája.
- Hatékonyan használható sok, osztott feldolgozási egységből álló hálózati rendszerek esetén.
- Egy új szerver könnyen hozzáadható a rendszerhez és integrálható annak többi részével.

30

Példa



Egy film és kép könyvtári rendszer architektúrája

31

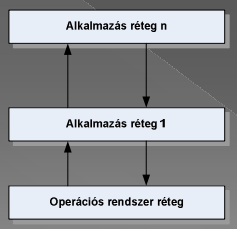
Rétegzett modell

- A rétegzett modell a rendszert rétegekbe szervezi.
- Amelyek mindegyike valamilyen szolgáltatásokat biztosít.
- Minden ilyen rétegre úgy tekinthetünk, mint egy *absztrakt gépre*
  - amelynek gépi nyelvét a réteg által biztosított szolgáltatások definiálják.
  - Ezt a „nyelvet", szolgáltatásokat használják az absztrakt gép következő szintjének megvalósítására.
  - Pl.: a hálózati protokollok OSI-referenciamodellje.

32

Rétegzett modell

- Rétegzett rendszerek:



33

Rétegzett modell

- A rétegalapú megközelítés segíti a rendszerek inkrementális fejlesztését.
- Mihelyt egy réteg kifejlesztésre került, a réteg által biztosított szolgáltatások egy része elérhetővé tehető a felhasználók számára.
- Egy réteg helyettesíthető egy másik,
  - azzal ekvivalens réteggel, ha interfésze nem változik meg.
- Egy réteg interfészének megváltozása:
  - vagy a réteg új lehetőségekkel történő bővítése csak a szomszédos réteget érinti.
- Csak a belső, gépfüggő rétegeket kell újraimplementálni ahhoz,
  - hogy figyelembe vegyük egy másik operációs rendszer vagy adatbázis lehetőségeit.

34

Rétegzett modell

- A megközelítés hátránya, hogy ily módon a rendszerek strukturálása igen bonyolulttá is válhat.
- Az alapvető adottságokat, amelyre minden absztrakt gépnek szüksége van a belső rétegek biztosíthatják.
- A külső réteg felhasználó által igényelt szolgáltatásainak „át kell ütniük" több szomszédos réteget is
  - ahhoz, hogy hozzáférjenek a több réteggel alattuk elhelyezkedő réteg szolgáltatásaihoz.
  - Ez felforgatja a modellt,
    - egy külső réteg a továbbiakban már nem csupán egyszerűen a közvetlen megelőzőjétől függ.

35

MODULÁRIS FELBONTÁS...

36

Moduláris felbontás

- A teljes rendszer-architektúra kiválasztásra után, dönteni kell az alrendszerek modulokra bontása során alkalmazott megközelítésről.
- Az alrendszerek és modulok nem különböztethetők meg egyértelműen,
  - de érdemes azokat a következő módon elképzelni:
  - Egy **alrendszer** egy olyan önálló rendszer, amely működése nem függ más alrendszerek szolgáltatásaitól. Az alrendszerek modulokból épülnek fel.
  - Egy **modul** olyan rendszerkomponens, amely más modulok számára szolgáltatás(oka)t biztosít.

37

Moduláris felbontás

- Az alrendszerek modulokra bontása során két fő stratégia ismeretes:
- ***1. Objektumorientált felbontással*** a rendszert egymással kommunikáló objektumok halmazára bontjuk fel.
- ***2. Funkcióorientált csővezetékek*** használatával a rendszert bemenő adatokat elfogadó és azokat kimenő adatokká alakító funkcionális modulokra bontjuk.

38

Köszönöm a figyelmet!

39