A SZOFTVERTECHOLÓGIA ALAPJAI

PPKE ITK, 2017. június 17.

Written by

Komporday András

2016

Tartalomjegyzék

1 A szoftvertervezés és más tervezési folyamatok összehasonlítása

A szoftverfejlesztési projekt különbözik más, hagyományos projektektől:

- A szoftver nem kézzelfogható.
- A termék egyedi, nem általánosítható.
- A szoftverfejlesztési projekt gyakran egyedi, nem általánosítható, mint a gépészeti, építési projektek.
- A szoftverfejlesztés folyamata nincs szabványosítva.
- Sok szoftver projekt egyedi.

1.1 Miért célszerű projektszervezetben végezni a szoftverfejlesztést?

- A **projektvezetés** feladata, hogy a szoftver a tervezett ütemezés szerint, határidőre, a követelményeknek megfelelően készüljön el.
- A **projektmenedzsmentre** azért van szükség, mert a szoftverfejlesztés mindig kötött pénzügyi és megszabott időkeretek között folyik, amelyeket a megrendelő, vagy a fejlesztő szervezet jelöl ki.

A szoftverfejlesztési projekt különbözik más, hagyományos projektektől. A szoftverfejlesztési projekt gyakran egyedi, nem általánosítható, mint a gépészeti, építési projektek. Ennek okai:

- A szoftver nem kézzelfogható.
- A termék egyedi, nem általánosítható.
- A szoftverfejlesztés folyamata nincs szabványosítva.

1.2 Miért és milyen gondot okoz a szoftverprojekt vezetője számára, hogy a szoftver nem látható, megfogható? Milyen módon lehet ezt a gondot csökkenteni?

Rizsa

1.3 Milyen típusú terveket kell készíteni egy projekt tervezésekor?

• A szoftverfolyamat dokumentumai – Cél: a szoftverfejlesztés támogatása, dokumentálása a választott folyamatmodell és technológia igényei szerint. • A szoftverprojekt dokumentumai – Cél: támogatni a szoftverfejlesztés, mint projekttevékenység tervezését, vezetését, követését, határidőre és a kívánt eredménnyel való befejezését. • A minőségbiztosítás dokumentumai – Cél: biztosítani és bizonyítani a projekt termékeinek (szoftver és dokumentáció) minőségét. • Projekt előkészítő dokumentumok – Javaslatok, előtanulmányok, stb. • A projekt végrehajtását támogató dokumentumok – Projektindító dokumentum – Jegyzőkönyvek (projektértekezletekről, megbeszélésekről, stb.) – Jelentések (a vezetésnek, a megrendelőnek) – Beszámolók (a vezetőknek, a projekt résztvevőinek) • A projekt lezárásának dokumentumai – Értékelések – Beszámolók

2 A szoftverfolyamat fázisai és modelljei, szoftverfejlesztési stratégiák

2.1 A szoftverfolyamat fázisai

A szoftverfolyamat a szoftver termék előállítására irányuló tevékenységek sora. Nincs egységes, minden szoftver kidolgozására alkalmas folyamat. Emberi és szervezeti tényezők is variálhatják. Az általános tevékenységek:

- Szoftverspecifikáció: a szoftver feladatainak és a megszorításoknak specifikációja
- Szoftverfejlesztés: a szoftver rendszer elkészítése
- Szoftvervalidáció: annak bizonyítása, hogy az elkészített rendszer a követelményeknek megfelelően működik
- Szoftverevolúció: a szoftver továbbfejlesztése a változó igényeknek megfelelően

2.2 A szoftverfolyamat modelljei, szoftverfejlesztési stratégiák

A szoftverfolyamat modellje a folyamat absztrakt reprezentációja. Általános folyamatmodellek:

Vízesés modell: az alapvető tevékenységek önálló fázisok. Jól áttekinthető, követhető fejlesztési folyamatot eredményez. A folyamat termékei szerződésekkel könnyen lefedhetők (specifikációs és tervezési dokumentumok, program(ok), stb.). Egymástól elkülönült fázisokra osztja a projektet (költséges egy korábbi fázisra visszatérni, pl. specifikációs vagy tervezési hiba esetén). Csak a projekt végén, átadáskor derülnek ki a specifikációs hibák. Nem képes rugalmasan alkalmazkodni a megrendelői igények változásaihoz. Csak a követelmények pontos ismeretében alkalmazható.

Fázisai:

1. Követelmények elemzése és meghatározása
2. Rendszer- és szoftvertervezés
3. Implementáció és egységteszt
4. Integráció és rendszerteszt
5. Működtetés és karbantartás

Evolúciós fejlesztés: Alapgondolata, hogy ki kell dolgozni egy kezdeti implementációt, amelyet a felhasználó véleményezhet és azt finomítani kell az elfogadásig. A specifikáció, fejlesztés és validáció összefonódik.

Feltáró fejlesztés : a követelmények feltárása lépésenként, a megrendelővel együttműködve történik, folyamatosan kiegészítve a rendszert új funkciókkal, részekkel.

Eldobható prototípus: „Deszkamodellek" készítése és átadása az ügyfélnek, a követelmények pontosabb feltárása érdekében.

Előnyök:
- kis vagy közepes interaktív rendszerek, esetleg nagy rendszerek felhasználói interfészének tervezésére alkalmas
- rövid életciklusú rendszerek esetén előnyös.

Hátrányok:
- A projekt előrehaladása nem követhető
- A rendszerek struktúrájával nem foglalkozik
- Speciális eszközöket és ismereteket igényel (pl. gyors modellező eszközök alkalmazása)

Formális rendszerfejlesztés: A vízesés modellhez hasonló, de a fejlesztés a követelmények matematikai modelljéből, formális transzformációval állítja elő a szoftvert, több transzformációs lépésen keresztül. Minden transzformáció során, lépésenként kell végrehajtani a tesztelést.

Előnyök:

1. Kritikus rendszerek esetén, ahol kulcskérdés a biztonság és megbízhatóság

2. A transzformáció és bizonyítás részben automatizálható.

Hátrányok:

1. Speciális szakértelmet igényel
2. Egy rendszer kölcsönhatásait (pl. felhasználói interfész) nehéz formálisan specifikálni.
3. Komplex, nagy rendszereknél ez a módszer sem eredményez jobb minőséget vagy költségmegtakarítást

Újrafelhasználás alapú fejlesztés: a rendszert már létező, újrafelhasználható komponensek integrálásával állítja elő. Jellemző rá a COTS (*Commercial off the Shelf*) termékek felhasználása.s gyakran beépül a korábban ismertetett folyamatokba.

Lépései:

1. A követelmények meghatározása
2. Komponens elemzés
3. Rendszertervezés újrafelhasználással
4. Fejlesztés és integráció
5. Rendszer validáció

3 A CASE eszközök, rendszerek és osztályozásuk

A CASE (*Computer Aided Software Engineering - számítógéppel támogatott szoftver-fejlesztés*) **eszközök** a szoftverfolyamat egyes lépéseit, pl. a terv konzisztenciájának ellenőrzését, program fordítást, a teszteredmények összehasonlítását támogató programok. Ezekből az eszközökből épülnek fel a teljes fázisokat (pl. specifikációs vagy tervezési fázist) támogató **eszközkészletek**. Egy szoftverfolyamatot nagyrészt, vagy teljesen lefedő eszközkészletgyűjteményt CASE rendszernek vagy környezetnek nevezünk. Ezek legtöbbször több, integrált eszközkészletből állnak.

3.1 Az eszközkészlet komponensei

- Diagramszerkesztő(k)
- Modell elemző és ellenőrző eszközök
- Adatbázis tároló és lekérdező eszközök
- Adatszótárak

- Jelentésgeneráló eszközök
- Űrlap definiáló eszközök
- Import/export eszközök
- Kódgenerátorok

3.2 Osztályozásuk

A fejlesztési életciklusban való elhelyezkedésük szerint:

Upper CASE , mely stratégiai tervezésre projektvezetésre szolgál

Middle CASE , mely a rendszerelemzési és tervezési fázisok munkáját támogatja

Lower CASE , mely egyszerűbb rendszerspecifikációk készítésére szolgál

Eszköztípus	**Példák**
Tervezői eszközök	PERT eszközök, becslési eszközök, táblázatkezelők
Szerkesztő eszközök	Text editor, diagramszerkesztők, szövegszerkesztők
Változtatáskezelő eszközök	Követelmény követhetőségi eszközök, változásvezérlő rendszerek
Konfigurációkezelő eszközök	Verziókezelő rendszerek, rendszerépítő eszközök
Prototípuskészítő eszközök	Nagyon magas szintű programnyelvek,felhasználói interfész generátorok
Módszertámogató eszközök	Tervszerkesztők, adatszótárak, kódgenerátorok
Nyelvi feldolgozó eszközök	Fordítók, értelmezők
Programelemző eszközök	Keresztreferencia generátorok, statikus elemzők, dinamikus elemzők
Tesztelő eszközök	Tesztadat generátorok, állomány összeállítók
Nyomkövető eszközök	Interaktív nyomkövető, belövő rendszerek
Dokumentációs eszközök	Arculattervező programok, képszerkesztők
Újratervezési eszközök	Kereszthivatkozási rendszerek, program újrastrukturáló rendszerek

1. táblázat. CASE-eszközök funkcionális osztályozása

4 A szoftverfejlesztési projekt jellemzői, ütemezése, a projekt kockázatai, a kockázatok kezelése

A kockázatkezelés a lehetséges kockázati tényezők azonosítását és a projektre gyakorolt hatásuk minimalizálására vonatkozó tervek készítését jelenti.

A kockázat típusai:

Projektkockázat: A projekt ütemtervét, vagy az erőforrásokat veszélyezteti,

Termékkockázat: A szoftver minőségét, vagy teljesítményét veszélyezteti

Üzleti kockázat: A szoftver beszerzését, vagy fejlesztését végző szervezetet veszélyezteti

5 A szoftverkövetelmények típusai, a követelmények elemzése

5.1 Funkcionális követelmények

- A rendszer által nyújtandó szolgáltatások leírása: hogyan reagáljon a rendszer az egyes bemenetekre, illetve mit tegyen egyes helyzetekben.
- A rendszer funkcióit, illetve szolgáltatásait tartalmazzák.
- A szoftver típusától, a várható felhasználástól és a felhasználóktól függenek.
- A felhasználói funkcionális követelmények általánosan írják le, hogy mit kell elvégeznie a rendszernek.
- A funkcionális rendszerkövetelmények részletesen írják le az egyes funkciók bemeneteit, kimeneteit, a kivételeket, stb.

5.2 Nem-funkcionális követelmények

- A rendszer funkcióira és szolgáltatásaira vonatkozó megszorítások, időbeli korlátok, a fejlesztési folyamatra vonatkozó korlátozások, szabványok.

- A rendszerfunkciókon kívüli követelmények, mint a megbízhatóság, válaszidő, tárigények, vagy az I/O eszközök tulajdonságaira, az interfészek adatformátumaira, stb. vonatkozó megszorítások.
- Ide tartoznak a fejlesztés módszereire, a minőségellenőrzésre, a fejlesztőeszközökre (CASE) vonatkozó követelmények, vagy a rendszeren kívüli (pl. jogi) megkötések is.
- Még a funkcionális követelményeknél is kritikusabbak lehetnek (pl. repülőgép irányítás – megbízhatóság)

A nem-funkcionális követelmények osztályozása

A termékre vonatkozó követelmények: A termék viselkedését határozzák meg (pl. sebesség, megbízhatóság, hordozhatóság, stb.)

Szervezeti követelmények: A megrendelő és a fejlesztő szervezete által támasztott szabályzatok és ügyrendek követelményei (módszertan, programozási nyelv, stb.)

Külső követelmények: A rendszeren és a fejlesztésen kívüli követelmények. (együttműködés más rendszerekkel, jogszabályi, etikai, stb.)

A követelmények elemzése és specifikálása költséges folyamat, és megtörténhet, hogy a vezetőknek ilyen hiányos információk mellett kell elkészíteniük a rendszer kifejlesztésére vonatkozó kiindulási költségbecslést. Ilyen esetekben gyakran alkalmazott stratégia, hogy a fejlesztő szervezet és a megrendelő a projekt költségében állapodik meg először, majd a megállapított fejlesztési költség által megszabott korlátok betartásával hoznak döntéseket a rendszer elvárt minimális funkcionalitásáról.

6 Tervezés újrafelhasználással, komponensek felhasználása

Kialakulásának oka az, hogy az objektumorientált fejlesztés nem támogatja az újrafelhasználást. A komponensek az objektumosztályoknál sokkal absztraktabbak és különálló szolgáltatásoknak tekinthetők. A komponensek szolgáltatásokat nyújtanak a rendszer számára, a végrehajtás helyétől és a megvalósítás nyelvétől függetlenül. A komponens forráskódja általában nem hozzáférhető, belső állapotai nem láthatóak.

6.1 Mi a komponens, milyen interfészei vannak?

A komponensek szolgáltatásokat nyújtanak a rendszer számára, a végrehajtás helyétől és a megvalósítás nyelvétől függetlenül. A komponensek mérete az egyszerű függvénytől a teljes alkalmazási rendszerig terjed.

- Egy komponens egy függetlenül végrehajtható program, amely egy vagy több végrehajtható objektumból áll.
- A komponensek interfészeit publikálják és minden interakció ezeken az interfészeken keresztül folyik. A komponens forráskódja általában nem hozzáférhető, belső állapotai nem láthatóak.

Szolgáltatott interfészek A komponens által szolgáltatott interfészek.

Szükséges interfészek Azok az interfészek, amelyeket a komponenst használó rendszernek, vagy környezetének kell biztosítania.

A komponens alapú fejlesztés beilleszthető a szabályos szoftverfolyamatba, ha beépítjük abba az újrafelhasználással kapcsolatos tevékenységeket:

- Komponensek specifikálása,
- Komponensek megtalálása,
- A tervek (esetleg a követelmények) módosítása a meglelt komponensek tulajdonságainak megfelelően. Ez az **alkalmazkodó újrafelhasználás**

6.2 Milyen hátrányai vannak az újrafelhasználható komponensekkel történő szoftverfejlesztésnek?

- **Az újrafelhasználhatóság és a használhatóság ellentmondása:** Minél általánosabb interfésszel rendelkezik, annál inkább újrafelhasználható, de annál bonyolultabb, vagyis kevésbé használható.
- Az újrafelhasználható komponensek fejlesztési költségei többszörösen meghaladják az egyszerű, specifikus komponensek költségeit. Ezt egyetlen projekt költségeiből nem lehet fedezni, ezért kialakultak olyan szoftverfejlesztő vállalatok, amelyek specializálódtak az újrafelhasználható komponensek fejlesztésére.
- Az általános komponensek kevésbé effektívek, több erőforrást használnak és végrehajtási idejük hosszabb, mint a specifikus komponenseké.

6.3 Mire szolgának az alkalmazási keretrendszerek, hogyan csoportosíthatók?

Definíció: A keretrendszer absztrakt és konkrét osztályok gyűjteményéből és a köztük lévő interfészekből álló alrendszer-terv. A keretrendszerek úgy implementálhatók, hogy a terv részeit komponensek hozzáadásával egészítjük ki. Általában viszonylag nagy, újrafelhasználható egységek, de nem önálló alkalmazások.

Az alkalmazások több keretrendszer integrálásával hozhatók létre. A keretrendszer általános struktúra, amely a konkrét alkalmazás létrehozásakor konkrét osztályokkal kibővíthető.

A keretrendszer kibővítése az alábbiakat jelenti:

- A kertrendszer absztrakt osztályainak kiegészítése konkrét osztályokkal.
- Műveletek hozzáadása, amelyek meghívhatók a keretrendszer által kezelt események bekövetkezésekor.

A keretrendszerek hátránya a bonyolultság. Sok időt igényel az effektív használatukhoz szükséges megismerésük.

A keretrendszerek fajtái:

- **A rendszer infrastruktúrájának keretrendszerei** a rendszer infrastrukturális alapjainak (kommunikáció, felhasználói felületek, stb.) fejlesztését támogatják.
- **Köztes integrációs keretrendszerek** a komponensek közti kommunikációt és információcserét támogató szabványok és osztályok. (Ilyen például a CORBA, JavaBean, COM, DCOM, stb.)
- **Vállalati alkalmazások keretrendszerei** az egyes speciális szakterületi alkalmazások fejlesztését támogatják. A szakterületi tudást tartalmazzák (pl. pénzügy, telekommunikáció).

7 A szoftverkövetelmények specifikálása, dokumentálása, formális specifikáció, interfészspecifikáció, viselkedésspecifikáció

7.1 Követelménytervezés

A követelmények a rendszer szolgáltatásainak és a megkötéseknek leírásai. A követelménytervezés feladata annak felmérése, hogy a rendszer majdani felhasználója (megrendelője) mit vár a szoftvertől, azoknak a körülményeknek meghatározása, amelyek a rendszer fejlesztését és működtetését befolyásolják.

A követelmények dokumentuma: A követelmények dokumentuma (System Requirements Specification – SRS) írja le, hogy mit várnak a tervezett rendszertől, vagyis mit kell megvalósítania a tervezőknek. A dokumentumnak nem a rendszer tervét kell tartalmaznia, hanem a követelmények definícióit és specifikációját vagyis azt, hogy mit kell tennie a rendszernek és nem azt, hogy hogyan.

Folyamata:

1. Megvalósíthatósági tanulmány
2. A követelmények feltárása és elemzése
3. A követelmények validálása
4. A követelmények kezelése, változáskezelés

7.2 A szoftverkövetelmények típusai

Felhasználói követelmények A rendszer szolgáltatásainak közérthető leírása, diagrammokkal, táblázatokkal, ábrákkal, a felhasználó számára.

Rendszerkövetelmények Strukturált dokumentum a rendszer szolgáltatásainak részletes leírásával (funkcionális specifikáció). **Ez a szerződés alapja.**

Szoftver specifikáció A szoftver követelmények részletes leírása a fejlesztők számára.

A követelményeket az olvasó számára egyértelműen, pontosan kell leírni. A pontatlan követelmény specifikáció félreértéseket eredményez. Például:

- *A felhasználó követelménye: „Minden dokumentum típushoz megfelelő megjelenítőt kell biztosítani"*
- *A fejlesztő értelmezheti úgy, hogy csak egy szöveges megjelenítőre van szükség, így az összetett dokumentumokat nem lehet olvasni.*

7.3.1. Felhasználói követelmények

A felhasználói követelményeket úgy kell megfogalmazni, hogy az informatikában járatlan felhasználó is megértse, ezért itt nem célszerű modelleket alkalmazni, hanem természetes nyelven, táblázatokkal és diagrammokkal kell a felhasználói követelményeket érthetővé tenni.

A természetes nyelv alkalmazásának nehézségei:

1. Az egyértelműség és pontosság hiánya
2. A követelmények keveredése
3. A követelmények ötvöződése

Javaslatok a felhasználói követelmények leírására:

1. Dolgozzunk ki egységes formátumot az összes követelmény leírására. Használjuk a nyelvet következetesen, a szükséges követelményeket a „kell", a kívánatos követelményeket pedig a „javallott" szóval jelölhetjük.
2. Készítsünk glosszáriumot a szövegben használt fogalmak és rövidítések magyarázatára.
3. A fontos részeket vizuálisan is emeljük ki a szövegből.
4. Kerüljük a számítógépes zsargon használatát.

7.3.2. Rendszerkövetelmények

A rendszerkövetelmények a felhasználói követelmények részletesebb és rendezett leírását adják. A rendszertervezés alapjául szolgálnak, tartalmazhatják a rendszer modelljeit. Sokszor a szerződéshez csatolják, ezért a rendszer teljes és konzisztens meghatározását kell tartalmazniuk. A rendszerkövetelmények leírják, hogy a rendszernek mit kell elvégeznie, majd a tervek határozzák meg, hogy hogyan tegye.

7.4 Formális követelményspecifikáció PDL használatával

A programleíró nyelvek (PDL – Program Description Language) olyan, programozási nyelvből származó, de rugalmasabb nyelvek, amelyekkel egyértelműbben leírhatók egyes követelmény típusok. Például:

- Amikor egy művelet tevékenységek sorozatából áll és fontos azok sorrendje, vagy
- Hardver és szoftver interfészeket kell specifikálni.

Hátrányai:

- A PDL nem alkalmas a szakterületi követelmények definiálására,
- A jelölések csak programozói ismeretekkel értelmezhetők,
- A követelmények leírása inkább egy tervezési specifikációhoz hasonlít és nem a rendszer megértését segítő modell.

7.5 Interfész specifikáció

Egy új rendszernek legtöbbször más, már meglévő rendszerekkel kell együttműködnie. Az interfészeket a követelmények között kell specifikálni.

Az interfészek típusai:

1. Procedurális interfészek – eljárások hívása egy másik (al)rendszerből
2. Adatszerkezetek, amelyek az egyik (al)rendszertől egy másikhoz kerülnek

3. Adatreprezentációk

A formális jelölések egyértelművé teszik az interfészek definícióját.

7.6 A követelmények teljessége, konzisztenciája

1. A követelményeket elvileg mindenre kiterjedően, és ellentmondásmentesen kell leírni.
2. A teljesség azt jelenti, hogy a felhasználó által igényelt összes szolgáltatásnak szerepelnie kell.
3. A leírás konzisztens, ha nincs konfliktus, vagy ellentmondás a rendszer szolgáltatásai között.
4. A nagyméretű, komplex rendszereknél ez a gyakorlatban megoldhatatlan.
5. Bonyolult rendszerekben ellentmondások lehetnek a nem-funkcionális követelmények között. Ezért a követelményeket fontosság szerint súlyozni kell.

7.7 A formális módszerek

A mérnöki tudományokban általánosan elfogadott a matematikai modellezés és elemzés alkalmazása a tervezési folyamatokban. A szoftvertervezés azonban még nem tudott kialakítani mindenki által használható módszereket, bár széles körben folynak a kutatások a szoftverminőség javítása érdekében.

A szoftvertechnológia területén a formális módszerek a szoftver matematikai reprezentációján és analízisén alapulnak. Ezen formális módszerek közé tartozik:

- A formális specifikáció,
- A specifikáció elemezése és bizonyítása,
- A transzformációs fejlesztés
- A program verifikálás

A formális módszerek - az előzetes várakozások ellenére – nem tudtak teret hódítani, mert:

1. Kialakultak többé-kevésbé sikeres módszerek, mint az OO tervezés, konfigurációkezelés, a strukturált progra-mozás, stb., amelyek javították a szoftver minőségét.

2. Újabban a szoftver gyors piacra kerülése fontosabb, mint a minőség. A gyors fejlesztési technikák nem illeszkednek a formális módszerekhez.

3. A formális módszerek csak korlátozottan alkalmaz-hatók például a felhasználói interfészek, felületek és munkafolyamatok specifikálására.

4. A formális módszerek nehezen, vagy egyáltalán nem alkalmazhatók nagy rendszerek esetén.

A formális módszereknek csak korlátozott felhasználási lehetőségei vannak. Alkalmazásuk kockázata és költsége sok esetben nagyobb, mint a várható előnyök.

Ugyanakkor azoknál a rendszereknél, amelyeket formális módszerekkel fejlesztettek, a hibaarány alacsonyabb volt.

Ezért elsősorban kritikus rendszerek fejlesztésénél alkalmazzák, ahol a rendszer igen magas validálási költségeihez és az esetleges hibákból eredő katasztrófa költségeihez képest még kifizetődő a használata.

7.7.1. Formális specifikáció

- A formális specifikáció egy matematikai jelölésrendszert alkalmaz, pontosan specifikált szótárral, szintaxissal és szemantikával.
- A specifikálás és a tervezés nagymértékben összefonódik egymással.
- Az architektúra-tervezés adhatja az alapot egy specifikáció struktúrájához.
- A szoftverspecifikáció folyamatának előrehaladásával az ügyfél befolyása csökken, a vállalkozó befolyása növekszik.

Formális specifikációs technikák:

Algebrai megközelítés: A rendszert műveletek és azok kapcsolatai alapján írja le.

Modell alapú megközelítés: A rendszert állapotmodellel specifikálja, amely halmazokból és sorozatokból álló matematikai konstrukciókat tartalmaz, a műveleteket pedig aszerint definiálja, ahogy azok a rendszer állapotát módosítják.

Még kevés eszköz készült a formális módszerek támogatására

A formális specifikáció alkalmazása A formális specifikáció a szoftverfejlesztés kezdeti szakaszában kíván nagyobb erőfeszítéseket. A követelmények alaposabb és részletesebb elemzése azzal jár, hogy kevesebb lesz a hiba a követelményspecifikációban. A következetlenségek és a hiányosságok felfedhetők és kijavíthatók a formális modellekkel. Ezért a követelmények későn felfedezett hibáiból eredő többletmunka lesz kevesebb.

A formális specifikációt nyílt interfészek, kommunikációs protokollok definiálására is alkalmazzák.

7.8 Interfészspecifikásió

A nagy rendszereket alrendszerekre bontják, amelyek között jól definiált interfészeket kell specifikálni. Az alrendszerek közti interfészek specifikálása teszi lehetővé, hogy az alrendszerek fejlesztése egymástól függetlenül történjen. Az interfészek absztrakt adattípusokkal, vagy objektum osztályokkal definiálhatók. A formális specifikáció algebrai megközelítése különösen alkalmas az interfészek pontos specifikálására.

7.9 Viselkedésspecifikáció

Az algebrai specifikációs technikák nehézséget okozhatnak, amikor az objektumok műveletei nem függetlenek annak állapotától. A formális specifikációk gyakorlati alkalmazásában jobban elterjedtek a modell alapú specifikációs módszerek.

A modell alapú specifikáció a rendszerspecifikációt a rendszer állapotmodelljeként fejezi ki. (Ilyen nyelvek többek között a VDM, a B és a Z). A Z a rendszereket halmazokkal és a halmazok közötti relációkkal modellezi. Kombinálja a formális és az informális leírásokat és grafikus ábrázolást alkalmaz. Különösen alkalmas interfészek és szoftver specifikálására (ISO szabvány lesz).

A Z sémák konzisztenciája

Az összes sémában konzisztensnek kell lenniük a predikátumoknak. Egy sémában nem lehet olyan állítás, ami ellentmond egy másik séma predikátumának. Az inkonzisztencia a követelmények ellentmondására hívja fel a figyelmet.

Az ellentmondásokat a rendszer implementálása előtt fel kell oldani.

8 A szoftverkövetelmények, a követelmények feltárásának, validálásának és kezelésének módszerei

A követelmények a rendszer szolgáltatásainak és a megkötéseknek leírásai. A követelménytervezés feladata annak felmérése, hogy a rendszer majdani felhasználója (megrendelője) mit vár a szoftvertől, azoknak a körülményeknek meghatározása, amelyek a rendszer fejlesztését és működtetését befolyásolják.

8.1 Követelmények feltárása és elemzése

Az informatikusok interjúkat készítenek és workshopokat tartanak a megrendelő kulcsembereivel (szakterületi képviselők, vezetők, és majdani felhasználók), hogy felderítsék, milyen szolgáltatásokat kell biztosítania a rendszernek.

A követelmények feltárása bonyolult, mert:

- A kulcsemberek gyakran nem tudják, hogy mit várhatnak és várnak egy számítógépes rendszertől.
- A kulcsemberek a saját szakterületük fogalmait használják, a követelménytervezőknek ezeket kell megérteniük.
- Az egyes szakterületeknek különböző elvárásai vannak.
- Egyes kulcsfigurák a saját pozíciójuk erősítésére akarják felhasználni az új rendszert.
- A környezet változása folyamatosan módosítja a követelményeket, a változásokat követni kell.

8.1.1. Nézőpont-orientált feltárás

Nagy rendszerek különböző felhasználói különböző nézőpontból látják a rendszer szolgáltatásait, eltérő követelményeik vannak, amelyek gyakran átfedik egymást, sokszor ellentmondanak egymásnak. A rendszerkövetelmények feltárását és elemzését tehát több perspektívából kell végezni, nincs egyetlen helyes út a probléma megközelítésére.

A nézőpont-orientált szemlélet felismeri a különböző perspektívákat és segít a különböző kulcsszereplők egymásnak ellentmondó követelményeinek felfedésében. A nézőpontok kívülről tekintik a rendszert, így strukturálják a követelmények feltárását. Főleg interaktív rendszerekhez alkalmas.

A nézőpontok típusai

Adatforrás vagy adatnyelő: Az adatok előállításának és feldolgozásának nézőpontjai. Az elemzés kiterjed az összes adatforrásra és adatnyelőre, továbbá a feldolgozások azonosítására és vizsgálatára.

Reprezentációs eszközkészlet: A különböző típusú rendszermodellek nézőpontjai. Ezek összehasonlításából kitűnik, melyek azok a követelmények, amelyeket az egyes rendszermodellek hibásan értelmeznek.

A szolgáltatások fogadója: A rendszer szolgáltatásait felhasználó (ember, másik rendszer, stb.) nézőpontjai.

Módszerek a követelményelemzéshez Céljuk a rendszerkövetelmények strukturált feltárá-sának, elemzésének, tervezésének támogatása.

Típusok:

Nézőpont orientált módszerek: (pl. VORD – Viewpoint-Oriented Requirements Definition)

Forgatókönyvek:
- Esemény forgatókönyvek
  - Leírják, hogyan használják a rendszert a gyakorlatban.
  - Használati esetek (Az UML-nél foglalkozunk vele).

Etnográfia: A rendszerek társadalmi, szervezeti környezete, az emberek munkavégzési módja felől közelíti a rendszerkövetelményeket. Gyakran kiegészül a prototípus-készítéssel.

8.2 A követelmények validálása

Feladata annak igazolása, hogy a követelmények a megrendelő kívánságainak megfelelő rendszert definiálják. A hibás, vagy hiányos követelmények nagy veszteségeket okoznak, ezért a validáció igen fontos:

> A hibás követelmények javítása a rendszer átadása után gyakran százszor annyiba kerül, mint egy implementációs (pl. programozási) hiba kijavítása.

Az érvényesség ellenőrzése: A felhasználó által előre nem látott funkciók feltárása.

Az ellentmondás-mentesség ellenőrzése: Ellentmondó megszorítások, vagy rendszerfunkciók kiszűrése.

A teljesség ellenőrzése: Annak ellenőrzése, hogy a dokumentum a felhasználó által kért összes funkciót tartalmazza-e.

A megvalósíthatóság ellenőrzése Az elképzelt rendszer megvalósítható-e a rendelkezésre álló technológiával, a tervezett idő alatt, az adott költséggel.

Az igazolhatóság ellenőrzése: A rendszerkövetelmények dokumentumai alkalmasak-e arra, hogy utólag igazolják, az átadott rendszer teljesíti a követelményeket.

8.3 A követelmények kezelése

A követelmények kezelése a követelmények változásának követésére, kézbentartására szolgáló folyamat.

A követelmények soha nem lehetnek teljesek és konzisztensek.

- A szoftverfolyamat során új követelmények merülnek fel, ahogy az üzleti környezet változik, és a feladat megértésében előbbre jutunk.
- A különböző nézőpontok különböző követelményeket támasztanak a rendszerrel szemben, amelyek gyakran ellentmondanak egymásnak.

A követelménykezelés során fel kell készülni a követelmények változására. Ehhez szükség van:

1. A követelmények egyedi azonosítására (ld. Specifikáció).
2. A változáskezelés folyamatának kidolgozására.
3. A követelmények és az összefüggések változásának követésére.
4. A változások és hatásuk elemzésére.

CASE eszközök támogathatják a változáskezelést.

9 A rendszermodellek típusai, környezeti és viselkedési modellek, adatmodell típusok

9.1 Rendszermodellek

A rendszermodellezés segíti az elemzőket a rendszer funkcionalitásának megértésében. Egyes modellek alkalmazhatók a felhasználóval folytatott kommunikációban is. A különböző modellek eltérő nézőpontból ábrázolják a rendszert:

- A **környezeti modellek** a rendszer környezetét és kapcsolatait mutatják be.
  - A rendszer határainak ábrázolására szolgálnak (mi tartozik a rendszerhez és mi nem)
  - A határok kijelölése gyakran nem technikai, hanem társadalmi, vagy szociális szempontoktól is függ.
  - A rendszer és külső rendszerekkel való kapcsolatainak ábrázolása az architekturális modell feladata.
  - A környezeti modell ábrázolási módja általában egyszerű blokkdiagram.
- A **folyamatmodell** a teljes munkafolyamatot bemutatja.
  - A folyamatmodell alapján lehet kijelölni, hogy a folyamat mely részeit kell támogatnia, vagy elvégeznie a számítógépes rendszernek.
  - A folyamatok és a folyamatok közti információ áramlás bemutatására adatfolyam modellek használhatók.
- A **viselkedési modellek** a rendszer átfogó viselkedésének leírására szolgálnak. Típusai:

  - **Adatfolyam modellek**: Bemutatják, hogyan dolgozza fel a rendszer az adatokat. Modellezik az adatfeldolgozást a rendszerben.
    * Azt mutatják be, hogyan áramlanak végig az adatok a feldolgozási lépések sorozatán és milyen átalakuláson mennek keresztül.
    * Segítik az elemzőket abban, hogy megértsék, mi történik a rendszerben.
    * Egyszerű jelölésrendszert alkalmaznak, ezért a megrendelő is könnyen megérti.
    * Funkcionális szempontból modellezik a rendszert, és alkalmasak a rendszer külső adatkapcsolatainak ábrázolására is.
  - **Állapotátmenet modellek**: Bemutatja, hogyan reagál a rendszer a különböző eseményekre. A rendszer viselkedését modellezik, a belső és külső eseményekre adott válaszokat írják le.
    * Gyakran használják valósidejű rendszerek modellezésére.
    * A rendszer állapotait csomópontként, az eseményeket nyilakkal jelöli. Egy esemény hatására a rendszer egyik állapotából egy másik állapotba kerül.
    * Az állapotdiagramok az UML jelölésrendszer részét képezik.
    * Feltételezi, hogy a rendszer egy adott időpontban a lehetséges állapotok egyikében van.
  - A rendszer viselkedésének leírásához mindkét típusra szükség van.

- A **szerkezeti modellek** a rendszer felépítését, illetve az adatok szerkezetét ábrázolják.

9.2 Adatmodellek

A nagy rendszerek sokféle adatot tárolnak és dolgoznak fel, nagyméretű adatbázisokat alkalmaznak. Az adatbázisok sok esetben a rendszertől függetlenül léteznek, máskor a rendszerrel együtt kell létrehozni azokat.

Az adatmodellek a rendszer által feldolgozott adatok logikai szerkezetének meghatározására szolgálnak. Az adatbázis tervezésben széles körben alkalmazzák őket. Leginkább elterjedt az **egyed-tulajdonság-kapcsolat** modellezése.

Az egyedeket műveletekkel nem rendelkező, egyszerűsített objektumosztályoknak tekinthetjük, így az UML osztálymodellje használható az adatok modellezésére is.

Az UML nem tartalmaz külön jelölésmódot az adatmodellezésre, az adatokat az objektumok és a köztük lévő kapcsolatok segítségével modellezi.

Az adatmodelleket gyakran adatfolyam-modellekkel együtt használják

9.2.1. Adatszótárak

Az egyed-típus-kapcsolat modelleket célszerű kiegészíteni adatszótárral. Az adatszótár tartalmazza az összes nevet, ami a modellben szerepel, az egyedleírásokat, kapcsolatokat és tulajdonságokat.

Előnyei:

1. Névkezelés, a névütközések kizárása,
2. Szervezeti információk tárolása, támogatja az elemzést, tervezést, implementációt és evolúciót.
3. Sok CASE eszköz tartalmazza az adatszótár támogatást.

9.2.2. A modell

A modell absztrakt leírása egy olyan rendszernek, amelynek követelményeit előzőleg már összegyűjtötték, rendszerbe foglalták és elemezték. Az absztrakt modell jellemzője, hogy részleteket hagy el, egyszerűsít. Kiemeli a lényeget. Vagyis rendszermodell nem egy másik reprezentációja a rendszernek.

9.2.3. A rendszermodellek típusai

Adatfeldolgozási modell: Adatfolyam diagramok, az adatok feldolgozását mutatják a rendszeren belül.

Kompozíciós modell: Egyed-kapcsolat diagramok. Bemutatják, hogyan épülnek fel az egyedek más egyedekből.

Architekturális modell: Az alrendszereket mutatják be, amelyekből a rendszer felépül.

Osztálymodell: Objektum osztály/öröklődési diagramok, az egyedek közös tulajdonságait ábrázolják.

Inger-válasz modell: Állapotátmenet diagramok, a rendszer belső és külső eseményekre adott reakcióit írják le.

10 Objektumorientált rendszermodellek, öröklődési modellek, az objektumok viselkedésének modellezése

A rendszerkövetelményeket (főleg interaktív rendszerek esetében) gyakran objektum-modellel írják le. Az objektummodell objektumosztályokkal modellezi a rendszert. Az objektummodellek természetes módon tükrözik az általuk manipulált valós világbeli egyedeket. Az elvont, magasabb szintű egyedeket ezzel a megközelítéssel nehezebb modellezni.

10.0.1. Objektumosztály

Közös tulajdonsággal rendelkező objektumok halmazának és az objektumok által nyújtott szolgáltatásoknak (műveleteknek) absztrakciója.

Jellemzői:

1. **Hasonló tulajdonságú** objektumok halmaza: Szerkezeti és viselkedésbeli jellemzők hasonlósága.
2. Az osztálynak **van neve**: A nevet az osztályba tartozó összes objektum örökli.
3. **Lehetnek attribútumai**, paraméterei: Hozzáférési módok: public, private, protected.
4. Tartoznak hozzá **szolgáltatások, műveletek**: Az osztály minden objektumára, vagy az osztály egészére vonatkozó műveletek.
5. Tartozhat hozzá **import felület**: Az általa igényelt szolgáltatások definíciói.
6. Rendelkezhet **megvalósítási résszel**: A megvalósítás leírása.
7. Van **látható és láthatatlan része**
8. Lehet **absztrakt vagy konkrét** osztály
9. Lehet **paraméteres (sablon)** osztály

10.0.2. Objektum

Az objektumok az objektumosztály példányai, végrehajtható egyedek, az objektumosztály tulajdonságaival és szolgáltatásaival.

Az objektumok jellemzői:

1. **Azonosítható**: Az objektumok egymástól megkülönböztethetők, függetlenül az állapotuktól.
2. **Tulajdonságok, attribútumok** tartoznak hozzá: Ezek lehetnek kötött, formális paraméterek is.
3. **Állapot** tartozik hozzá: Az attribútumok konkrét értékei az objektum mindenkori állapotát határozzák meg.
4. **Műveletek** tartoznak hozzá: Ezek lehetnek leképezések, tevékenységek, események.
5. **Korlátozott láthatósággal** rendelkezik: van látható része (export és import műveletek) és van láthatatlan része (az ábrázolás és a szolgáltatások megvalósításának részletei).

Egy adott szakterület egyedeinek objektumosztályba sorolása több rendszerben is felhasználható.

Néhány objektummodell fajta:

- Öröklődési modell
- Aggregációs modell
- Viselkedési modell

A rendszerek felhasználói számára a funkcionális modellek (pl. adatfolyam diagram) könnyebben érthetőek, mint az objektummodellek.

10.1 Öröklődési modell

Az objektumosztályokat taxonómiába szervezi. A taxonómia olyan osztályozási séma, amely megmutatja, egy osztály hogyan kapcsolódik más osz-

tályokhoz, közös tulajdonságokon és szolgáltatásokon keresztül. Az osztályhierarchia tervezése az egyik legnehezebb feladat, mivel az egyes ágakon kerülni kell a duplikálást.

Az alacsonyabb szinten lévő osztályok: öröklik a magasabb szintű osztályoktól tulajdonságaikat és szolgáltatásaikat rendelkezhetnek speciális tulajdonságokkal és szolgáltatásokkal is.

Objektumaggregáció

Egyes objektumok más objektumokból épülnek fel, azok aggregátumaként. Az aggregációs modell azt mutatja meg, hogy hogyan keletkezik több osztályból egy aggregált osztály.

10.2 Viselkedési modell

Az objektumok viselkedése az általuk biztosított műveletek sorrendjének ábrázolásával történhet (szekvencia diagram). A szekvencia diagram voltaképpen egy forgatókönyv, amely a használati eseten alapul.

A szekvencia diagramok mellett az UML-ben együttműködési diagramokat is használunk, ahol az objektumok által váltott üzenetek sorozatát ábrázoljuk.

11 Szoftverprototípus készítése, a prototípusok fajtái, prototípuskészítés és adatbázis programozás

11.1 Rendszerprototípus készítése

A **prototípuskészítés** a követelménytervezés része, a követelmények feltárásának és validációjának eszköze.

A **prototípus** a szoftverrendszer kezdeti verziója, amely alkalmas a rendszer koncepciójának bemutatására és kipróbálására. Korábban úgy tekintették, hogy a prototípus alacsonyabb rendű a kívánt rendszernél. Ma a prototípus- és a normál rendszer közti határ fokozatosan elmosódik és sok rendszert az evolúciós modell alapján készítenek.

11.1.1. A prototípusok alkalmazása

A felhasználó nem látja előre, hogyan fogja használni az új rendszert. A prototípus elsődleges célja az, hogy segítse a felhasználókat a rendszerkövetelmények megértésében:

A követelmények feltárása: a prototípussal a felhasználók megtapasztalhatják, hogyan fogja a rendszer a munkájukat támogatni.

A követelmények validálása: a prototípus felfedheti a hibákat és hiányosságokat a követelményekben.

11.1.2. A prototípuskészítés előnyei

- Felfedi a szoftver felhasználója és készítője közti félreértéseket. Csökkenti a követelményekkel kapcsolatos kockázatokat.
- Kiderülhet, hogy hiányzik valamely szolgáltatás, vagy ellentmondások vannak a szolgáltatások között.
- A szoftverfolyamat elején már egy – legalábbis részben - működő rendszer áll rendelkezésre.
- A prototípus felhasználható a rendszer-specifikáció alapjaként.
- Támogathatja a felhasználók képzését és a rendszertesztet.

11.1.3. A prototípuskészítés folyamata

1. Prototípusfeladatok megállapítása - Prototípuskészítési terv
2. Prototípus funkcióinak meghatározása - Vázlat kidolgozása
3. Prototípus fejlesztése - Futtatható prototípus
4. Prototípus kiértékelése - Kiértékelési jelentés

11.1.4. Előnyei és hátrányai

Előnyök:

- A rendszer használhatóbb lesz.

- A rendszer jobban illeszkedik a felhasználó igényeihez.
- Javul a tervezés minősége.
- Gyorsabban elkészül a rendszer.
- A fejlesztéshez kevesebb erőforrásra van szükség (költségcsökkentés).

Veszélyek:

- Eldobható prototípust végleges rendszerként használnak (teljesítmény, funkcionalitás, megbízhatóság)
- A gyors fejlesztésből és az iterációból fakadó hibák: válaszidő, rendszerstruktúra

11.1.5. A prototípuskészítés helye a szoftverfolyamatban

Evolúciós prototípus készítése: Célja egy működő rendszer átadása a megrendelőnek. A legfontosabb követelmények implementálásával egyszerű rendszer készül, amelyet újabb követelmények feltárásával fokozatosan egészítenek ki új funkciókkal. Weblap fejlesztésben és e-business alkalmazásokban használják.

Eldobható prototípus készítése: Célja a rendszerkövetelmények feltárása és validálása. A nem teljesen megértett követelmények megvalósítása és bemutatása segíti a feltárást. A követelményspecifikáció elkészülte után nem használható fel.

11.2 Evolúciós prototípuskészítés

Olyan rendszereknél célszerű alkalmazni, ahol nem készíthető el előre a specifikáció. Ilyenek általában az intenzív felhasználói interfész-használatot igénylő rendszerek.

Nincs részletes rendszerspecifikáció, sokszor a részletes követelménydokumentum is hiányzik. s A fejlesztéshez gyors, iterálható fejlesztő eszközökre és módszerekre van szükség. Mivel nem készül követelményspecifikáció, a validáció is csak a rendszer bemutatásával történhet.

11.2.1. Az evolúciós prototípuskészítés folyamata

1. Absztrekt specifikáció készítése
2. Prototítpus építése
3. Prototípus használata
4. Ha megfelelő a rendszer: a rendszer átadása
5. Egyébként: Új Prototípus építése

11.2.2. Az evolúciós prototípuskészítés jellemzői

- A specifikáció, a tervezés és az implementáció átlapolható.
- A rendszer inkrementumok sorozataként fejlődik, és kerül a felhasználóhoz, vagyis a felhasználó kulcsfigurái minden inkrementum tervezésében és értékelésében részt vesznek.
- Gyors fejlesztő eszközök és technikák alkalmazhatók (CASE eszközök, 4GL, modellező nyelvek: BPML-Business Process Modeling Language).
- A felhasználói felületek GUI fejlesztő eszközökkel készíthetők.

11.2.3. Előnyei

Felgyorsul a rendszerfejlesztés. A gyors fejlesztés, az új rendszer sürgős használatba vétele gyakran fontosabb mint a követelmények részletes feltárása, vagy a hosszú távú karbantarthatóság.

Növelhető a felhasználó elkötelezettsége: A felhasználók bevonása a rendszerfolyamatba azt eredményezi, hogy a rendszer nagyobb valószínűséggel felel meg az elvárásoknak, és a használatba vételkor a felhasználók már ismerik azt és tudják alkalmazni.

11.2.4. Hátrányai

Vezetési problémák A jelenlegi vezetési módszerek a vízesés modellre alkalmazhatók. Az új technológiák alkalmazásához speciális ismeretekre, esetleg más munkatársakra van szükség.

Karbantartási problémák A folytonos változások a prototípus szerkezetének sérülését okozhatják, a dokumentáció hiánya és a speciális fejlesztő eszközök a karbantartást veszélyeztetik.

Szerződéskötési problémák A fix áras szerződéshez előre ismerni kell a rendszer vázlatos követelményeit és tervét. A ráfordítás alapú szerződést pedig a megrendelő nem fogadja el.

11.2.5. A prototípus, mint specifikáció

Egyes követelményeket (mint pl. a biztonság-kritikus funkciókat) nem lehet a prototípusba beépíteni, így nem fognak szerepelni a specifikációban. Egy implementáció nem lehet egy szerződés jogi melléklete. A nemfunkcionális követelmények nem tesztelhetők teljes mértékben.

11.3 Eldobható prototípus készítése

Célja a követelményspecifikációból fakadó kockázat csökkentése. A prototípust egy kezdeti specifikáció alapján készítik, svalidálásra átadják a felhasználónak, majd eldobják. Az eldobható prototípus nem tekinthető végleges rendszernek, mert:

1. Több rendszertulajdonság kimaradhat a prototípusból,
2. Nem készül specifikáció a hosszú távú karbantartásra.
3. A prototípus még nem a megfelelő struktúra szerint épül.

11.3.1. A prototípuskészítés folyamata

1. A követelmények körvonalazása
2. Prototípusfejlesztés
3. Prototípus kiértékelése
4. Prototípus elfogadása vagy új prototípus fejlesztése
5. Rendszerspecifikáció készítése
6. Szoftverfejlesztés *(újrafelhasználható komponensekkel)*
7. A rendszer validálása: elfogadás vagy további fejlesztés

8. A szoftver átadása

11.3.2. A prototípus átadása

A vezetők gyakran nyomást gyakorolnak a fejlesztőkre, hogy egy működő eldobható prototípust végleges rendszerként adjanak át. Ez nagyon veszélyes, mert:

- A prototípus nem alakítható úgy, hogy a nem-funkcionális követelményeknek (teljesítmény, megbízhatóság, stb.) eleget tegyen.
- A prototípus rendszerint dokumentálatlan marad, mert a cél a gyors elkészítés és bemutatás.
- A változtatások miatt a rendszer struktúrája általában romlik a fejlesztés során.
- A prototípus készítésekor az általános szervezeti szabványokat nem tartják be (minőségbiztosítás, technológiai fegyelem, projekt dokumentálás).

11.4 Gyors prototípuskészítési technikák

A gyors prototípuskészítéshez az alábbi technikák alkalmazhatók:

1. Fejlesztés dinamikus, magas szintű nyelven,
2. Adatbázis-programozás,
3. Komponensek és alkalmazások összeépítése.

A gyakorlatban ezeket együttesen alkalmazzák.

A legtöbb prototípuskészítő eszköz tartalmazza a vizuális programozás támogatását, ahol grafikus szimbólumok reprezentálják a függvényeket, adatokat, feldolgozó szkrtipteket. Az eszköz a rendszer vizuális reprezentációjából generálja a végrehajtható programot.

11.4.1. Fejlesztés magas szintű nyelven

Olyan nyelvek, amelyek hatékony futási idejű adatkezelő eszközöket tartalmaznak. Korábban a nagy rendszerek fejlesztéséhez nem használtak ilye-

neket, mert nagy teljesítményű futtató rendszereket igényelnek, ami növeli a tárigényt, csökkenti a futási teljesítményt.

Némely nyelv olyan integrált támogató környezetet tartalmaz, amely felhasználható a gyors prototípuskészítéshez. A magas szintű nyelvek többsége fejlett felhasználói interfész fejlesztő képességekkel rendelkezik.

11.4.2. Prototípuskészítő nyelvek

Szempontok az alkalmas nyelv kiválasztásához:

Az alkalmazási terület jellege: Természetes nyelvű feldolgozáshoz a Lisp, vagy a Prolog alkalmasabb.

A felhasználói interakció jellege: Az intenzív web alapú felhasználói interakció kidolgozására a Java, vagy Smalltalk több eszközt kínál.

A támogatási környezet: A nyelvvel együtt sokféle eszköz és komponens segíti a prototípuskészítést.

A rendszerprototípus különböző részei különböző nyelven programozhatók. A különböző nyelven írt részek közötti kapcsolatot kommunikációs keretrendszer teremtheti meg.

Nagy rendszerek prototípusának készítésekor nincs egyetlen ideális nyelv. A rendszer egyes részei az igényeknek megfelelő leginkább alkalmas nyelven készülhetnek. Mivel a nyelvek eltérő egyedfogalmakat használhatnak, a részek közti adat- és vezérlési kapcsolatok sok addicionális kódot igényelhetnek.

11.5 Adatbázis-programozás

Az evolúciós fejlesztés az adatbázison alapuló kis-, közepes üzleti alkalmazások területén általánosan alkalmazott technika. A kereskedelmi adatbáziskezelő rendszerek olyan 4GL fejlesztő eszközöket tartalmaznak, amelyek támogatják a lekérdezést/ datkezelést (SQL), táblázatkezelést, jelentés generálást, felhasználói felületek tervezését, stb.

Az adatfeldolgozási alkalmazásokban sok közös jegy van: adatbázis manipulációk (keresés, frissítés, rendezés, stb.), egyszerű műveletek, űrlapkezelés, stb. Egy 4GL-ben ezeket általánosítják. Gyakran integrálhatók CASE eszközökkel is. Ezek generálhatnak SQL-t, vagy alacsonyabb szintű kódot.

Negyedik generációs nyelvek (4GL)

Tulajdonságok:

- Interaktív, gyakran grafikus űrlapgenerálás,
- Űrlapkezelés (strukturált űrlapok összekapcsolása, mezőellenőrzés)
- Web-es kapcsolatok (böngésző)
- Nagy tár- és erőforrás igény

Elemei:

- Adatbázis-kezelő rendszer, ez köti össze az összes többi elemet
- Adatbázis-programozási nyelv
- Interfész-generátor
- Táblázatkezelő
- Jelentésgenerátor

Hátrány: Szabványok hiánya (gyártó-specifikus)

11.6 Prototípuskészítés újrafelhasználással

Az alkalmazás szintjén: Teljes alkalmazási rendszerek integrálása a prototípusba úgy, hogy egymás funkcióit megosztva használják. (Például szabványos szövegszerkesztő alkalmazása a szövegszerkesztésre.)

A komponensek szintjén: Az egyedi komponensek integrálása egy szabványos keretrendszerbe. (Például egy szkriptnyelv, mint a Visual Basic, vagy olyan integrációs keretrendszer, amely CORBA, szabványon alapul, vagy .net, JavaBean futtatásra alkalmas.)

11.7 Felhasználói felületek prototípusai

A felhasználót be kell vonni a felhasználói felületek tervezésébe, a fejlesztő nem erőltetheti rá saját elképzeléseit. A prototípusok készítése segít a felhasználók bevonásában. A felhasználói interfész fejlesztése a rendszerfejlesztési költségek növekvő hányadát adja.

Az interfész-generátorok segítenek abban, hogy a felhasználók gyorsan véleményt alkossanak a felületekről. A felületek specifikációjából jól strukturált programot generálnak. A web alapú felhasználói interfészek készítésére jól ismert weblap tervező eszközök léteznek.

12 Architekturális tervezés, a rendszer strukturálása, rendszermodellek, vezérlési modellek

12.1 Architekturális tervezés

A rendszertervezés folyamatának kezdeti lépcsőfoka. Az architekturális tervezés az a tervezési folyamat, amelynek során kijelölik a rendszert alkotó alrendszereket és azt a keretrendszert, amely vezérli az alrendszereket és biztosítja közöttük a kommunikációt. A folyamat végeredménye a szoftver architektúra, amely a tervezés alapjául szolgál.

Az architekturális tervezés célja a rendszer együttműködő alrendszerekké való felbontása.Az architektúra terv általában egyszerű blokkdiagram formájában ábrázolja a rendszer (mindenki által megérthető) struktúráját. Részletesebb modellek is alkalmazhatók, amelyek megmutatják:

1. Hogyan osztják meg egymás közt az alrendszerek az adatokat,
2. Hogyan kommunikálnak egymással.

12.1.1. Feladata

Összekötni a specifikáció és a tervezés folyamatát. Kialakítani a rendszer alapvető struktúráját és azt a keretrendszert, amely a rendszert egységbe foglalja és működését irányítja.

Gyakran egyes specifikációs tevékenységgekkel párhuzamosan végezhető. Magába foglalja a fő rendszerkomponensek és azok vezérlésének, valamint kommunikációjának meghatározását.

12.1.2. A rendszerstruktúra meghatározása

A bonyolult rendszerek egymással lazán összefüggő részfeladatokból állnak, amelyek önállóan végrehajthatók, de egymással vezérlési és adatcsere kapcsolatban állnak. Példa: banki szolgáltató rendszer alrendszerei:

Központi feladatok: Ügyfélnyilvántartás, könyvelés, számlavezetés, betétkezelés, hitelkezelés, kártyakezelés, vezetői információs rendszer, stb.

Ügyfélkiszolgálással kapcsolatos feladatok:

- Ügyfél tranzakciók: Személyes kiszolgálás a bankfiókban, telefonos-, Internetes tranzakciók, kártyás vásárlások, ATM, pénzforgalom, hitelezés, értékpapír forgalom, stb. (egyéni- és vállalati ügyfelek számára)
- Bankközi tranzakciók: Átutalások, hitelek, fedezet-igazolás, értékpapírműveletek, stb.

12.1.3. A jól megtervezett architektúra előnyei

A tervezői megbeszélések alapját képezi. A tervezés kulcsszereplői számára érthetővé teszi a rendszer vázát.

Támogatja a kritikus kérdések korai elemzését. Az architektúra terv alapján megítélhető, hogy a rendszer eleget fog-e tenni olyan kritikus követelményeknek, mint a teljesítmény, megbízhatóság, karbantarthatóság, skálázhatóság.

Megalapozza az újrafelhasználhatóságot. Az alrendszerekre bontás és azok fő tulajdonságainak meghatározása lehetőséget ad újrafelhasználható komponensek kifejlesztésére (vagy felhasználására), termékcsaládok kidolgozására, amelyben az azonos feladatokat újrafelhasználható komponensek oldják meg.

12.1.4. Az architekturális tervezés tevékenységei

1. **A rendszer strukturálása:** A rendszert több alrendszerre bontjuk és azonosítjuk a kommunikációs igényeket az alrendszerek között.

2. **A vezérlés modellezése:** Általános modell készül a rendszer részei közötti vezérlési kapcsolatokról.

3. **Moduláris felbontás:** Az azonosított alrendszerek modulokra bontása és a modulok közti kapcsolatok azonosítása.

Az architekturális tervezés tevékenységei többnyire nem szekvenciálisan, hanem páthozamosan folynak.

12.2 Alrendszerek és modulok

Alrendszer: Az alrendszer olyan – szolgáltatásaik alapján – egységként kezelhető komponensek rendszere, amely önállóan oldja meg feladatát. Modulokból, vagy más alrendszerekből áll, szabványos interfészen keresztül veheti igénybe más alrendszerek szolgáltatásait.

Modul Olyan rendszer-komponens, amely szolgáltatás(oka)t nyújt más moduloknak és igénybe veszi mások szolgáltatásait, de nem tekinthető független alrendszernek. Más, egyszerűbb modulokból (komponensekből) áll.

12.3 Architektúra modellek

Az architekturális tervezés során architektúra modellek készülnek, amelyek különböző nézőpontokból ábrázolják a rendszer architektúráját:

Statikus szerkezeti modell: A különálló alrendszereket és rendszerkomponenseket ábrázolja.

Dinamikus folyamatmodell: Megmutatja, hogy a rendszer hogyan szerveződik folyamatokba működése alatt.

Interfészmodell: Az alrendszerek közötti interfészeket ábrázolja.

Kapcsolatmodell: Az alrendszerek közti adatfolyammal mutatja be a kapcsolatokat.

Architekturális stílusok

Egy rendszer architekturális modellje legtöbbször valamilyen általános modellezési stílus szerint készül. Ilyen stílusok alkalmazása egyszerűbbé és

egységesebbé teszi a rendszer architektúra definiálását. A heterogén, nagy rendszerek architektúrája azonban nem ábrázolható egységes stílusban. Az eltérő funkciójú részek eltérő modellezést kívánnak. A tervezőknek kell megtalálniuk a feladatra leginkább alkalmas modellezési stílust.

12.4 Az architektúra és a követelmények

A rendszer architektúrája kihat a nem-funkcionális rendszerkövetelmények kielégítésére, így meghatározza a:

Teljesítményt: Egy rendszer teljesítménye jobb lesz, ha nagyméretű modulokból áll, mert kevesebb kommunikáció zajlik a modulok között.

A védelmet: A jobb védelem érdekében rétegezett szerkezetet célszerű alkalmazni, a kritikus rendszerelemeket a legbelső rétegben elhelyezve.

A biztonságot: A biztonsággal kapcsolatos műveletek egy, vagy néhány alrendszerben legyenek.

A rendelkezésre állást: Redundáns komponensek alkalmazásával növelhető.

A karbantarthatóságot: Sok önálló, könnyen változtatható komponensből kell felépülnie.

12.5 A rendszer strukturálása

A rendszer együttműködő alrendszerekké való felbontása. Az architektúra terv általában egyszerű blokkdiagram formájában ábrázolja a rendszer (mindenki által megérthető) struktúráját. Részletesebb modellek is alkalmazhatók, amelyek megmutatják, hogy

- hogyan osztják meg egymás közt az alrendszerek az adatokat, illetve
- hogyan kommunikálnak egymással.

12.6 Vezérlési modellek

A strukturális rendszermodellek az alrendszerekre való felbontást ábrázolják, nem tartalmaznak vezérlési információkat. A vezérlési modellek az alrendszerek közötti vezérlési folyamatokat modellezik.

12.6.1. Központosított vezérlés:

Egy alrendszer végzi a teljes rendszer vezérlését, indítja, leállítja, stb. a többi alrendszert.

Hívás-visszatérés modell: Fa-struktúrájú modell, ahol a csúcson van a vezérlő alrendszer. A vezérlés hívások sorozatán keresztül jut el a modulokhoz. Szekvenciális rendszerekhez alkalmazható (pl. listafeldolgozás, listázás, jelentésgenerálás).

Kezelő modell: Konkurens rendszerek modellezésére alkalmas. Egy központi rendszerkomponens koordinálja, indítja, állítja le a rendszerfolyamatokat (komponenseket, vagy alrend-szereket), amelyek párhuzamosan is végrehajthatók. Alkalmazható szekvenciális rendszerekben is, ahol a vezérlő modul állapotváltozók értéke alapján hívja meg az egyes alrendszereket.

12.6.2. Eseményalapú vezérlés:

Minden alrendszer reagálhat az őt érintő külső vagy más alrendszer által generált eseményekre. A környezet által generált események irányítják a rendszert. Az esemény nemcsak bináris jel, hanem érték változása is lehet. Az esemény időzítése az eseményt feldolgozó alrendszer hatályán kívül esik.

Eseményvezérelt rendszer lehet pl. egy táblázatkezelő is, ahol egy cella értékének megváltozása más cellákat is megváltoztat, vagy más alrendszert aktivizál.

Broadcast modell: Az eseményről mindegyik alrendszer értesül, és az reagál rá, amelyiknek ez a feladata.

Megszakításvezérelt modell: Valós idejű rendszerek modellje, ahol egy megszakítás-kezelő észleli az eseményt és elindítja az esemény feldolgozásáért felelős alrendszert.

13 Objektumorientált tervezés, UML diagramok, az objektuminterfész specifikáció

13.1 Az objektumorientált tervezés

Az objektumorientált fejlesztés az alábbi – összefüggő, de különálló - fázisokból áll:

1. **Objektumorientált elemzés (OOA)**: Az alkalmazás objektumorientált modelljének kidolgozása.
2. **Objektumorientált tervezés (OOD)**: A követelményeknek megfelelő szoftverrendszer objektumorientált modelljének kidolgozása.
3. **Objektumorientált programozás (OOP)**: A szoftverterv objektumorientált programnyelven történő megvalósítása.

Az egyes fázisok között nincs explicit határvonal, egy következő lépés az előző finomításával jár.

13.1.1. Az objektumorientált tervezés lényege

- Az objektumok a valódi világ vagy egy rendszer elemeinek absztrakciói, amelyek karbantartják saját állapotukat.
- Az objektumok függetlenek, de együttműködnek egymással, „elrejhetik" állapotukat és jellemzőiket.
- A rendszer funkcionalitása az objektumok szolgáltatásaiban fejezhető ki.
- Nem használnak megosztott adatterületeket, üzenetek útján kommunikálnak egymással.
- Az objektumok szétoszthatók, szekvenciálisan, vagy párhuzamosan végrehajthatók.

13.1.2. Az objektumorientált tervezés előnyei

- Könnyebb a szoftvert karbantartani: Az objektumok önálló egységekként értelmezhetők.
- Az egyes rendszerekben egyértelmű leképezés van a való világ elemei és a rendszer objektumai között.

- Az objektumok megfelelő újrafelhasználható komponensek.
- A gyakran használt elemekre léteznek objektum-könyvtárak,
- A tervezési minták a gyakran előforduló struktúrákra általános, és nyelv független mintákat adnak.

13.1.3. Objektumok és objektumosztályok

Az objektumok: egy szoftver rendszer vagy a valódi világ elemeinek reprezentációi.

Az osztály: a hasonló tulajdonságú objektumok egy halmaza. Az osztályok közötti kapcsolatokat **relációknak** nevezzük.

13.2 Az UML modellezési nyelv

Az objektumorientált tervezés során, az utóbbi 20 év során kidolgozott jelölések egységesítésével létrejött modellezési nyelv. Jelölésrendszerével az objektumorientált analízis és tervezés során készíthető modellek ábrázolását támogatja. Az objektumorientált tervezésben de-facto szabvánnyá vált.

Asszociáció az UML jelöléseivel

Az objektumok és objektumosztályok kapcsolatban vannak más objektumokkal és objektumosztályokkal. Az UML-ben az asszociációkat az objektumosztályok közötti vonallal és a kapcsolatot leíró megjegyzésekkel modellezik.

Az asszociációk általánosak, de jelezhetik, hogy:

- egy objektum egy attribútuma egy vele kapcsolatban álló objektumtól, vagy
- egy objektum egy műveletének implementációja egy vele kapcsolatban lévő objektumtól függ.

13.3 Az objektuminterfész specifikációja

Az objektumok interfészen keresztül kommunikálnak egymással, üzeneteket küldve és fogadva. Ez a gyakran eljáráshívással és paraméterek átadásával valósul meg, amikor a szolgáltatást kérő az alábbiakat adja meg:

- Név = eljárás név
- Információ = paraméterlista

Az üzenet tartalma:

1. A kért szolgáltatás neve,
2. A szolgáltatás végrehajtásához szükséges információ és a szolgáltatás eredményt kérő neve.

13.3.1. A kommunikáció típusai

• **Szinkron végrehajtás:** A hívó objektum megvárja a szolgáltatás befejeződését. Az fentebb ismertetett eljáráshívás szinkron végrehajtást jelent.

Párhuzamos végrehajtás: Ha az objektumok konkurens szálakként vannak implementálva, a hívó tovább folytatja működését. Ilyen esetekben – és az osztott rendszerekben – az objektumok kommunikációja (sokszor szöveges, pl. XML) üzenetek formájában valósul meg.

13.3.2. Objektumosztályok közötti kapcsolatok

Relációk:

Asszociáció : Kétirányú társítás két osztály között. (Konkrét esetben: összekapcsolás, absztrakt esetben társítás)

Aggregáció : Az osztály objektumainak egymáshoz rendelése.

Kompozíció : Egy osztály objektumai a másik osztály objektumait fizikailag tartalmazzák.

Öröklődés : Egy általános osztályból származtatással létrehozott speciális(abb) osztály jön létre.

Generalizáció és öröklődés Az öröklődés (generalizáció) olyan absztrakciós mechanizmus, amely az entitások osztályozására használható. Az osztályok hierarchiába szervezhetők, ahol egy osztálytól (szülőosztály) egy- vagy több osztály (leszármazott osztály) örökli a szülőosztály attribútumait és műveleteit. A hierarchiában alacsonyabb szinten lévő osztályok **öröklik a szülőosztály attribútumait és műveleteit és újakkal egészíthetik ki azokat**, sőt meg is változtathatják szülőosztályaik némelyik attribútumát, vagy műveletét.

Az öröklődés előnyei:

- Lehetővé teszi az újrafelhasználhatóságot a tervezés és a programozás szintjén egyaránt.
- Az öröklődési diagramban ábrázolhatók a szakterülettel és a rendszerrel kapcsolatos szervezeti ismeretek.

Az öröklődéssel kapcsolatos problémák:

- Az objektumosztályok nem önállóak. Nem érthetők és értelmezhetők a szülőosztályok ismerete nélkül.
- A tervezők a tervezés során újra felhasználhatják az elemzés során készült öröklődési diagramokat, amivel munkát takarítanak meg, azonban nagy mértékben csökkenhet a modell hatásfoka.
- Az elemzés, tervezés és az implementáció során készített öröklődési diagramoknak más a feladata, ezért külön kell elkészíteni és karbantartani azokat.

Mivel az öröklődés az OOD (objektumorientált tervezés) alapvető eszköze, ezzel kapcsolatban többféle megközelítés alakult ki:

1. Az öröklődési hierarchia azonosítása az OOD alapvető feladata. Az implementáció pedig nyilvánvalóan egy objektumorientált programozási nyelv feladata.
2. Az öröklődés az implementáció hasznos eszköze, amely segíti az attribútumok és a műveletek újrafelhasználását. Az öröklődési hierarchiát azonban nem célszerű a tervezési fázisban meghatározni, mert ezzel túl sok megkötést viszünk be az implementációba.

Az öröklődés olyan fokú bonyolultságot eredményezhet, amelyet kritikus rendszerekben célszerű elkerülni.

13.3.3. Objektumorientált tervezési folyamat

Lépései:

1. Definiáljuk a rendszer összefüggéseit és használatának módjait.
2. Tervezzük meg a rendszerarchitektúrát.
3. Azonosítsuk a rendszer legfontosabb objektumait
4. Dolgozzuk ki a tervezési modelleket
5. Határozzuk meg az objektumok interfészeit

14 Felhasználói felületek tervezése, alapelvek, megjelenítés, felhasználói támogatás

A felhasználó a kezelőfelületen keresztül kerül kapcsolatba a rendszerrel, ennek alapján alkot véleményt, csak ezután ismeri meg a rendszer funkcionalitását. A rosszul tervezett kezelőfelület gyakran katasztrofális hibákhoz vezet. A szegényes, vagy következetlen felhasználói kezelőfelület sok rendszer bukásához vezetett.

Nagy fejlesztő szervezetekben szakértőket alkalmaznak (grafikus, pszichológus, szakterületi szakértő), de kis/közepes cégeknél a kezelőfelület megtervezése is a szoftver tervező feladata.

14.1 Grafikus felületek

A korai rendszerek csak alfanumerikus terminálokat alkalmazhattak, a kezelőfelület karakteres, vagy űrlap jellegű volt. Már ekkor kialakultak a kezelőfelületekkel szembeni alapkövetelmények:

1. Legyen strukturált, következetes, áttekinthető,
2. Biztosítson segítő szolgáltatásokat,
3. A hibákat egyértelműen jelezze.

Ma csaknem minden rendszer nagyfelbontású, színes, grafikus felületet támogat. Az interakcióra nemcsak a klaviatúra, hanem egér, vagy más kijelölő eszköz is rendelkezésre áll.

14.1.1. A grafikus kezelőfelület előnyei

- Könnyebben megtanulható és használható, akár számítógépes ismeretek nélkül is.
- A felhasználó több képernyőt használhat az interakcióra, gyorsan válthat különböző alkalmazások között, az információ látható maradhat az éppen nem aktív ablakban is.
- A felhasználó a teljes képernyő bármely részét elérheti, ez gyors interakciót tesz lehetővé.

14.1.2. A felhasználócentrikus centrikus tervezés

A sok egyéb mellett a szoftvertervező feladata a felhasználói kezelőfelület tervezése is. A felhasználó központú kezelőfelület-tervezés megköveteli, hogy a tervező

1. Alaposan megismerje a felhasználó tevékenységét (a munkafolyamatot, amelyet a rendszernek támogatnia kell) és felkészültségét,
2. A felhasználót kezdettől bevonjuk a tervezés folyamatába,
3. Először papíron, a struktúra felvázolásával, majd prototípusok készítésével tegyük számára megfoghatóvá és érthetővé a tervet.

14.2 A felhasználói kezelőfelületek tervezésének alapelvei

A kezelőfelület tervezésekor figyelembe kell venni a felhasználók igényeit, gyakorlatát és képességeit. Az emberek fizikai és mentális képességei korlátozottak (rövid távú memória), a felhasználói felület tervezésekor ezt figyelembe kell venni. A grafikus felhasználói felületek tervezésének alapelvei minden felhasználói interakció tervezésének alapjául szolgálhatnak.

14.2.1. Tervezési alapelvek

1. A felhasználói jártasság figyelembevétele
2. A felületnek olyan kifejezéseket és fogalmakat kell használnia, amelyeket az átlagos felhasználó ismer.
3. A felület konzisztenciája

4. A menüknek és parancsoknak ugyanazzal a formátummal kell rendelkezniük, hasonló műveleteket hasonló módon kell megvalósítani.
5. Minimális meglepetés

A felhasználóban kialakul egy modell a rendszer működéséről. A hasonló tevékenységeknek hasonló hatást kell kiváltaniuk, különben a rendszer kellemetlen meglepetéseket okoz felhasználó számára.

Visszaállíthatóság Minden helyzetben számítani kel arra, hogy a felhasználó hibázhat, ezért gondoskodni kell arról, hogy a hibát kijavíthassa:

- Visszavonási lehetőség (undo), esetleg többszintű,
- Veszélyes tevékenységek megerősítése (pl. törlés),
- „Puha törlés"

14.2.2. A felhasználó és a rendszer kapcsolata

Az interaktív rendszer tervezésekor két kulcskérdést kell megoldani:

1. Hogyan jusson el az információ a felhasználótól a rendszerhez, és
2. Hogyan jusson el az információ a rendszertől a felhasználóhoz.

A felhasználói beavatkozás és az információ megjelenítése egy összefüggő keretrendszerbe integrálható, amely biztosíthatja a konzisztenciát és a felhasználói támogatást.

14.3 Az interakciók fajtái

14.3.1. Közvetlen manipuláció:

A felhasználó közvetlenül a képernyőn látható objektumot kezeli (pl. törléshez kukába viszi).

Előnyei:

1. Könnyen tanulható és gyors,

2. A felhasználó azonnal visszajelzést kap, így a tévedés gyorsan visszavonható.

Hátrányai:

1. Bonyolult lehet a felhasználó tevékenységéről (szándékáról) a megfelelő információt begyűjteni a program számára,
2. Csak akkor használható, ha a feladatok és objektumok egyértelműen megkülönböztethető ikonokkal reprezentálhatók.

14.3.2. Menükiválasztás:

A felhasználó a rendszer által felkínált (sokszor helyzet-függő) listából választhat, a kijelölést egér, vagy kurzormozgatással, rövidített név beírással is végezheti. Alkalmazható az egyszerű (pl. érintőképernyős) terminálokon is.

Előnyei:

1. A felhasználónak nem kell parancsokat megjegyeznie,
2. Kevés gépelést igényel és a hibák könnyen kivédhetők,
3. Állapotfüggő súgó alkalmazható.

Hátrányai:

1. Az akciók közötti logikai összefüggések (and, or) nem jeleníthetők meg,
2. Kevés választási lehetőséget enged meg, a sok lehetőséghez strukturálni kell a menüket.
3. A gyakorlott felhasználó számára lassú.

Űrlapkitöltés: Az űrlap az aktuális állapothoz alkalmazható. Olyan rendszerekben alkalmazzák, ahol sok adatot kell bevinni (pl. adatrögzítés).

Előnyei:

1. A felhasználói hibák felfedhetők és jelezhetők, illetve kivédhetők,
2. Könnyen megtanulható.

Hátránya:

1. Nagy képernyőfelületet foglal

14.3.3. Parancsnyelv:

A felhasználó parancsokat gépelve utasítja a rendszert (pl. Unix)

Előnyei:

1. Egyszerű, olcsó terminálon is alkalmazható,
2. Egyszerűen feldolgozható (pl. fordító technikával)
3. Bonyolult, egymásba ágyazott parancsok is kezelhetők,
4. Rugalmas.

Hátrányai:

1. Nehezen tanulható, az átlagos felhasználó számára bonyolult,
2. Gépelési gyakorlatot kíván
3. A hibakezelést (hibajelzés, visszavonás) nehéz megoldani
4. A parancsnyelveket a gyakorlott felhasználó számára lehet alkalmazni. A menürendszer alternatívájaként célszerű alkalmazni.

Az **alkalmi felhasználók** több támogatást, a **gyakorlott felhasználók** egyszerűbb, gyorsabb működést várnak.

14.3.4. Természetes nyelv:

A felhasználó a parancsokat természetes nyelven gépeli be, amelynek szótára korlátozott. Az ilyen rendszerek általában speciális alkalmazási területet szolgálnak ki. A természetes nyelv megfelelő az alkalmi felhasználó számára de a gyakorlott felhasználó nem kedveli a túl sok gépelés miatt.

14.3.5. Többszörös felhasználói interfészek

Az eseti és a gyakorlott felhasználók számára külön felületet célszerű megvalósítani (pl. Model-View-Controller, MVC)

14.4 Az információ megjelenítése

A rendszer megjeleníti a felhasználó számára közlendő információkat. Ez az információ megjelenhet közvetlenül szöveges formában, vagy más módon (pl. grafikusan, akár hang kíséretében). A jól tervezett rendszerekben maga az információ és az azt megjelenítő szoftver különválik.

A Model-View-Controller (MVC) általánosan alkalmazott architektúra az adatok többféle megjelenítésére. Az MVC paradigmát a Smalltalkban dolgozták ki, de azóta általánosan elterjedt az interaktív rendszerek grafikus felhasználói kezelőfelületének tervezésében. Lényege, hogy különválasztja a az információt (az üzleti logikát), a megjelenítés vezérlését és a megjelenítést.

Az információ lehet:

Statikus információ: Értéket kap a munkafázis (session) kezdetén és ez a session ideje alatt nem változik meg, lehet numerikus, vagy szöveges

Dinamikus információ: Megváltozik a munkafázis alatt és a megváltozott értéket a felhasználó számára meg kell jeleníteni, lehet numerikus, vagy szöveges

A megjelenítés stílusában meg kell különböztetni őket.

14.4.1. A megjelenítés módjának kiválasztása

Szempontok:

- A felhasználónak pontos információra van-e szüksége (numerikus), vagy

különböző adatok közti kapcsolatok, arányok érdeklik (grafikus)?

- Milyen gyorsan változik az információ? (A gyorsan változó információt grafikusan, vagy többféle módon kell megjeleníteni.)
- Azonnal szükség van-e rá?
- Egy változást követően be kell-e avatkoznia a felhasználónak valamilyen akcióval? (Ha igen, a megváltozott információt ki kel emelni.)
- Szükség van-e közvetlen beavatkozási felületre?(Ha igen, az információ közelében kell erre lehetőséget adni.)
- Szöveges vagy numerikus a megjelenítendő információ? Fontosak-e a relatív értékek? (Ha igen, grafikus.)

14.4.2. Analóg és digitális megjelenítés

Digitális megjelenítés:

- pontos értékeket közöl,
- Kevés helyet foglal a képernyőn.

Analóg megjelenítés:

- Egy pillantással áttekinthető,
- Relatív értékeket is képes közölni:
  - Egy állandó értékhez képest (egy határhoz közeli értéket színnel még külön ki lehet emelni), vagy
  - Korábbi minimális-maximális értékhez képest

14.4.3. Figyelmeztető szöveg megjelenítése

A figyelmeztetés megjelenítésekor a grafika kiemeli a fontos szöveget, az információ jellegére ikonnal is utalhatunk. A szöveg és grafika mellett hang is használható a figyelem felkeltésére, amennyiben feltételezhető, hogy a felhasználók nagy része rendelkezik hangkártyával.

14.4.4. Hibaüzenetek

A hibaüzenetek tervezése különösen fontos: a kezdő felhasználó ezekkel találkozik a leggyakrabban. A rossz, vagy számára érthetetlen hibaüzenetek miatt elutasíthatja a rendszert. Az üzeneteknek udvariasnak, előrevivőnek és következetesnek kell lennie. A felhasználó háttere, gyakorlata a hibaüzenetek tervezésének meghatározó tényezője.

14.5 A felhasználó támogatása

A felületnek könnyen elérhető segítő, vagy súgó rendszerrel kell rendelkeznie. A súgót strukturálni kell, nem szabad túl sok információt közölni. Előnyös a helyzetfüggő súgó alkalmazása.

A felhasználó támogatása kiterjed a rendszer minden megjelenési formájára: súgó, hibaüzenetek, kézikönyvek, stb. A felhasználó tájékoztatását be kell építeni a felhasználói felületekbe, hogy minden helyzetben kérhessen támogatást, vagy kapjon információt, ha hibát vétett.

> Célszerű a súgó és az üzenő rendszert összeépíteni, hogy minden üzenetről magyarázatot kérhessen a felhasználó.

14.5.1. A súgó tervezése

A felhasználó segítségért, információért fordul a súgóhoz. A súgó tervezésekor mindkét igényt figyelembe kell venni. Többféle lehetőséget kell biztosítani, ehhez **több belépési pontra** van szükség.

A jó súgórendszer hierarchikus szerkezetű, de bonyolult hálós struktúrájú, ahol az információs egységek között sokféle kapcsolat van. Több ablak alkalmazásával érthetővé tehető a bonyolult hierarchia.

Szempontok:

- Több belépési pontra van szükség, hogy a felhasználó a rendszer különböző állapotaiból léphessen be.
- Ugyanakkor hasznos azt jelezni, hogy éppen hol jár a súgó hierarchiájában.

- Célszerű a korábban bejárt útvonalat is megjeleníteni, mert a bonyolult hálóban könnyen elvész a felhasználó. Ez a visszalépéseket is támogathatja.

14.5.2. A súgó információtartalma

A súgó nem lehet egy online kézikönyv! A képernyő nem felel meg a papírlapoknak. Az emberek másként olvassák a képernyőt, mint a papírt. A megjelenítés dinamikus természete segíti az információ megjelenítését.

A súgórendszer szövegeit az alkalmazást és a szakterületet jól ismerő embereknek kell megfogalmaznia.

14.5.3. Felhasználói dokumentáció

Az on-line súgó mellett papíralapú dokumentációt is kell készíteni a rendszerhez. A dokumentációnak a kezdőtől a gyakorlott felhasználóig mindenkit figyelembe kell vennie. A különböző csoportba tartozó felhasználók számára legalább ötféle dokumentumot kell készíteni.

Dokumentumtípusok:

Funkcionális leírás : A rendszer funkcióinak rövid leírása.

Bevezető kézikönyv : A rendszer helyes használatának leírása, sok példával.

Referencia kézikönyv : A rendszer lehetőségei, hibaüzenetek és teendők hiba esetén, minden esetre kiterjedően.

Telepítési dokumentum : A telepítés menete, a teendők listája, a beállítások ismertetése.

Üzemeltetési-, adminisztrátori kézikönyv : A rendszer működtetésének, a hibák kijavításának leírása.

15 Osztott rendszerek architektúrái, többprocesszoros architektúrák, kliens-szerver architektúrák

A hálózatok terjedésével lassan minden rendszer (még a beágyazott rendszerek is) más rendszerekkel kapcsolatban működik.

15.1 Az osztott rendszerek jellemzői:

- **Erőforrásmegosztás** (oda kell fordulni, ahol létezik a kívánt szolgáltatás : WebServices!)
- Konkurencia (többféle hw/sw szállító termékeit tartalmazzák)
- **Konkurencia** (az egyes gépekben párhuzamos folyamatok mennek végbe, amelyek időnként kommunikálnak és szinkronizálják egymást)
- **Skálázhatóság**
- **Hibatűrés**
- **Átlátszóság** (a felhasználó nem látja, hogy osztott rendszerrel van kapcsolatban) de esetenként szükség van arra, hogy a felhasználó tisztában legyen vele, honnan vesz igénybe erőforrásokat, szolgáltatásokat (pl. web-es alkalmazások nagy része)

15.1.1. Hátrányai

- **Bonyolultság**
  - nehezebb a rendszer és tulajdonságinak (pl. teljesítmény) tervezése,
  - karbantartási nehézségek, stb.
- **Kezelhetőség**: a különböző hardver és operációs rendszer operálása nagy nehézségeket okozhat.
- **Biztonság**: az osztott rendszer biztonságát sokszor szintén elosztva kell megoldani. (segítenek a modern rendszerek, pl. SSO (single-sign-on) megoldásai)

15.1.2. Tervezési kérdések

Erőforrások azonosítása : Névkonvenciókra van szükség, hogy megtalálhatók és hivatkozhatók legyenek az erőforrások (pl. interneten URL)

Kommunikáció: Az internet, TCP/IP sok mindenre megfelelő, de néha speciális kommunikációs protokollokra van szükség (valósidejű közvetlen kapcsolatok)

A szolgáltatás minősége: Sok tényezőtől függ (hw, op.rendszer/ek, architektúra: erőforrások elosztása, hálózat, a rendszer rugalmassága)

Szoftverarchitektúra: A funkciók elosztása a rendszer logikai komponensei között, ezek eloszlása a hardver erőforrások között (pl. adatbázis szerver önmagában többprocesszoros rendszeren futhat.) A logikai komponensek között köztes szolgáltatásra (middleware) van szükség.

15.1.3. Többprocesszoros architekrtúrák

A legegyszerűbb osztott rendszermodell: a különböző folyamatok külön processzorokon futnak. Példa: ipari folyamatirányítás

15.2 Tárolási modell

Az alrendszerek két módon cserélhetnek információt egymással:

1. A megosztott adatok egy központi adatbázisban vannak, amelyet minden alrendszer elérhet. Ez a tárolási modell (repository).
2. Minden alrendszernek van saját adatbázisa, és az alrendszerek üzenetek formájában cserélnek adatokat. A nagy adatmennyiséggel dolgozó rendszerek legtöbbször osztott adatbázis köré szervezett alrendszerekkel dolgoznak. Ilyenek például a nagy, vállalatirányítási rendszerek, CASE és CAD rendszerek, stb.

15.2.1. Megosztott tárolók alkalmazása

Előnyök

- Nagy tömegű adat esetén hatékonyabb, mert nem kell explicit módon átvinni az adatokat egyik alrendszerből a másikba.
- Az alrendszereknek nem kell foglalkozniuk azzal, hogyan keletkeztek az adatok.
- A védelem, biztonsági mentések, hozzáférés szabályozása, a visszaállítás, stb. központi funkcióként oldható meg.
- Tárolási sémán keresztül publikálható a megosztottság modellje (új alrendszerek integrálhatók, ha a modell megfelelő)

Hátrányok

- Az alrendszereknek közös – kompromisszumos - adatmodellt kell használniuk (teljesítmény)
- Az alrendszereknek törődniük kell azzal, hogy a többi alrendszer hogyan fogja használni az adatokat.
- Az egyes alrendszerek eltérő követelményeket támasztanak a védelem, helyreállítás, stb. közös funkciókkal szemben (pl. tranzakciók visszagörgetése)
- Nagyon bonyolult lehet az adatbázis elosztása több gép között. A nagy adatbáziskezelő rendszerek tartalmaznak eszközöket a megosztásra, ezek azonban nagy erőforrásokat igényelnek.

15.3 Osztott rendszerek architektúrái

15.3.1. Absztrakt gép modell (réteges modell)

Az alrendszerek közti interfészek modellezésére használják. Rétegekbe (absztrakt gépekbe) szervezi a rendszert, amelyek mindegyike adott szolgáltatásokat végez. Támogatja az egyes alrendszerek inkrementális fejlesztését. Az egyes rétegek egyszerűen kicserélhetők, csak az interfészek szabályait kell betartani, de annak változtatásához is csak a két szomszédos réteget kell módosítani.

Előnye, hogy mivel a hardvert, operációs rendszert a belső rétegekbe zárja, könnyen adaptálható különböző platformokra (protokoll modellek:ISO-OSI).

Hátránya: strukturálása bonyolult, egy külső réteg csak a közbensőkön keresztül férhet hozzá a legbelsőkhöz.

15.4 Kliens-szerver architektúra

Olyan osztott rendszermodell, amely bemutatja hogyan oszlanak meg az adatok és a feldolgozások a komponensek között. Jellemzője, hogy a szerverek általában maguk kezelik az adataikat.

Szerverek: Pl. Adatkezelő szerverek, nyomtatószerverek kommunikációs szerverek, stb.

Kliensek: Többnyire önálló alrendszerek, amelyek hozzáférnek a szerverek szolgáltatásaihoz. Egyszerre sok példányban futnak. Fajtái:

- Vékony kliens (böngésző, szkriptekkel)
- Vastag kliens (komplett kis alrendszer, helyi adatokkal)

Hálózat: A klienseknek biztosít hozzáférést a szerverek szolgáltatásaihoz.

Előnyök:

- Jól strukturált osztott architektúra.
- Könnyen kiegészíthető új szerverrel (új funkcióval).
- Alacsonyabb hardver követelményei vannak.

Hátrányok:

- Nincs megosztott, közös adatmodell, mindegyik alrendszer a saját szempontjai miatt kialakított adatmodellt használja (ez előny a teljesítmény szempontjából).
- Redundáns adatkezelés folyik minden szerverben.
- Nincs központi név- és szolgáltatás nyilvántartás, nehéz megtalálni, hogy milyen szerverek és szolgáltatások léteznek.

16 Verifikáció és validáció, a verifikáció tervezése, verifikációs és validációs módszerek

A V&V célja: megbizonyosodni arról, hogy a szoftver rendszer megfelel a céljának. (Vagyis nem az, hogy hibamentes!)

Verifikáció: Annak ellenőrzése, hogy valóban a megfelelő terméket készítjük el, vagyis, hogy a szoftver megfelel a specifikációnak.

Validáció: Annak bizonyítása, hogy a terméket jól készítjük el, vagyis hogy a szoftver valóban a megrendelő elvárásainak megfelelően működik (esetleg a specifikációval ellentétesen). A szoftvernek azt kell megvalósítania, amit a felhasználó valóban elvár tőle.

A verifikáció és validáció (V& V) folyamata a szoftver teljes életciklusára kiterjed, a szoftver folyamat minden fázisában szerepet kap. Alapvetően két célja van:

1. Felfedni a rendszerben rejlő hibákat
2. Meggyőződni arról, hogy a rendszer egy-egy konkrét működési szituációban használhatóan működik.

A V& V folyamatban kétféle technika alkalmazható:

Szoftver-átvizsgálás (inspekció): A rendszer reprezentációjának elemzése (Köv. Spec., Tervek, grafikus ábrázolások, forráskód). A forráskód elemzése automatizálható.

Szoftvertesztelés: A szoftver implementációjának tesztadatokkal való futtatása és a viselkedés megfigyelése (dinamikus verifikáció)

16.1 Programtesztelés

Még ma is a legelterjedtebb validációs technika (bár a szoftverfolyamat végén helyezkedik el). A hiba meglétét kell felfedeznie nem a hiba hiányát. Az a sikeres teszt, amely legalább egy hibát felfedez. Az egyetlen módszer a nemfunkcionális követelmények validálására. A statikus verifikációval együtt kell alkalmazni.

Az elfogadás szintje különböző célú rendszereknél különböző. Ezt befolyásolja:

- **A szoftver funkciója**: (biztonsági rendszer, prototípus)
- **A felhasználó elvárásai** (olcsó szoftver – több hiba)
- **Piaci környezet** (árak, versenytársak)
- **Kritikus rendszerek** (ne okozzon tragikus eseményeket)

Alapos tervezésre van szükség, hogy a legtöbb eredményt kapjuk az egyébként igen költséges tesztelésből és felülvizsgálatból. A V& V tervezését a fejlesztési folyamat elején meg kell kezdeni. A tervnek meg kell határoznia az arányokat a statikus verifikáció és a tesztelés között.

A teszt-tervezésre a nagyobb cégeknél általános szabványokat, szabályokat dolgoznak ki. Ennek alapján kell megtervezni és végrehajtani a termék konkrét tesztelését, és a tesztek dokumentálását.

16.2.1. A szoftver teszterv struktúrája

A tesztelési folyamat A fő tesztfázisok leírása.

A követelmények nyomon követhetősége Minden követelményt külön kell tesztelni.

A tesztelt elemek A tesztelendő szoftver termékek listája.

A tesztelés ütemezése A szoftverfejlesztési projekt részeként.

tesztek dokumentálása A tesztelés utólagos ellenőrzésére (minőségbiztosítás).

A tesztek hardver és szoftver követelményei A teszteléshez szükséges erőforrások.

Megszorítások A tesztelést gátló tényezők.

16.2.2. Inspekció (átvizsgálás)

A szoftver átvizsgálás célja a hiányosságok felderítése, a költséges tesztelés helyett a hibák kb. 60%-a felfedhető az átvizsgálás során. A fejlesztési folyamat kezdetétől alkalmazható, a dokumentumok (követelmények, tervek) átvizsgálásával. Egy átvizsgálás során több hiányosság felfedezhető, amíg egy teszt többnyire egy hibát fed fel. A tapasztalt vizsgálók (inspektorok) már ismerik és könnyen megtalálják a típushibákat.

Az inspekció és a tesztelés nem helyettesítik egymást, de a korai fázistól rendszeresen végzett átvizsgálás sok költséges tesztet előzhet meg. **Mindkettőt alkalmazni kell a V& V folyamatban.**

Az inspekció alkalmas eszköz arra, hogy ellenőrizze, megfelel-e a program a specifikációnak. A nem-funkcionális rendszerkövetelmények vizsgálatára

azonban a felülvizsgálat nem használható.

16.2.3. Cleanroom folyamat

A szoftverhibák elkerülését, nem pedig megtalálását és kijavítását célzó szigorú átvizsgálási folyamat. (A név a félvezető-gyártásból származik). A rendszer komponenseinek tesztelését helyettesíti átvizsgálásokkal, megfelelnek-e a specifikációnak. Inkrementális fejlesztési módszer, először a kritikus inkrementumokat szállítja le.

A Cleanroom jellemzői:

- Formális specifikáció (állapotátmenet modell, strukturált programozás, csak néhány vezérlési és adatabsztrakciós konstrukció használható)
- Inkrementális fejlesztés
- Statikus verifikáció (szigorú átvizsgálások)
- A rendszer statisztikai tesztelése

A Cleanroom folyamat szervezete

Specifikációs csapat: A rendszerspecifikáció kidolgozását és karbantartását végzi.

Fejlesztő csapat: A fejlesztést és verifikálást végzi. A szoftvert nem futtatja.

Hitelesítő csapat: A formális specifikáción alapuló statisztikai teszteket dolgozza ki (a fejlesztéssel párhuzamosan) és futtatja le.

17 Szoftvertesztelés, a hiányosságok tesztelése, tesztelés és belövés, integrációs tesztelés

17.1 Mi a programtesztelés feladata? Milyen alaptípusai vannak?

A szoftver implementációjának tesztadatokkal való futtatása és a viselkedés megfigyelése (dinamikus verifikáció)

Hiányosságok tesztelése: Feladata a rendszer hibáinak és hiányosságainak felfedése. Fajtái:

Komponens tesztek: Fekete doboz, ekvivalencia-osztályok, struktúrateszt, útvonal-teszt

Integrációs tesztek: „fentről lefelé/lentről felfelé", interfészteszt, stressztesztek

Objektumorientált tesztelés:

Interaktív rendszer esetén tesztelni kell:

- A menükön elérhető összes funkciót,
- Egyazon menüponton elérhető valamennyi rendszerfunkciót,
- A felhasználói inputok által használt összes függvényt, helyes és helytelen input adatokkal egyaránt.

Statisztikai tesztelés: a rendszer teljesítményének és megbízhatóságának tesztelése, valós helyzetekben (valós felhasználói inputtal és gyakorisággal).

17.2 Mi a különbség a tesztelés és a belövés között? Melyiknek mi a célja?

A hiányosságok tesztelése és a belövés különböző folyamatok:

- A verifikáció és validáció feladata a hibák, hiányosságok létezésének felfedezése.
- A belövés ezen hibák helyének lokalizálása és kijavítása.
- A belövés a program viselkedésére vonatkozó feltételezések felállításával kezdődik, majd ezen feltételezések vizsgálatával próbálja megtalálni a hibákat

17.3 Mi a teszteset és a tesztadat? Hogyan lehet a tesztadatok számát csökkenteni?

A tesztesetek a teszthez szükséges inputok és a vélt outputok specifikációi,

A tesztadatok a rendszer tesztelésére kidolgozott input adatok

17.4 Mit jelent a tesztadatok ekvivalencia-osztályozása? Írjon példát az ekvivalencia-osztályok alkalmazására.

Ekvivalencia-osztály: a rendszer input és output adatait valamilyen közös jellegzetesség szerint csoportosítják, amelyekre a rendszer hasonló módon reagál: *Például*: ha az input 5 jegyű valós szám 10.000 és 99.000 között, akkor az ekvivalencia-osztályok: $< 10.000, 10.000 \breve{} 99.000, > 99.999$ A fejlesztők legtöbbször az inputok tipikus értékeit veszik figyelembe. A teszteseteket a határértékek közelében és az osztályok közepéből célszerű kiválasztani: $\breve{} 00000, 09999, 10000, 99999, 10001$

Csak teljes körű tesztelés bizonyíthatná, hogy a rendszer hibamentes, de a teljes tesztelés lehetetlen. A teszteknek a rendszer a képességeit kell vizsgálniuk, nem a komponenseket. A fejlesztés során a rendszer régi képességeinek tesztelése fontosabb, mint az újonnan hozzáadott képességeké. A tipikus helyzetek tesztelése fontosabb, mint a határesetek tesztelése.

17.5 Mi a „fekete doboz" és a „fehér doboz" tesztelési stratégia lényege? Melyiket milyen esetben lehet alkalmazni?

A fekete doboz tesztelés

- Funkcionális tesztelésnek is nevezik.
- A programot fekete doboznak tekintjük, a tesztesetek a programspecifikáció alapján készülnek.
- Nem foglalkozik a program implementációjával.
- A tesztek tervezése a szoftverfolyamat korai szakaszában megkezdődhet.
- Az előreláthatóan hibát okozó tesztesetek tervezéséhez szakterületi ismeretekre van szükség.

Struktúrateszt

- Fehér doboz vagy üvegdoboz tesztelésnek is nevezik, mert a tesztek a program struktúrájának, implementációjának ismeretében készülnek.
- A struktúra és a kód ismeretében újabb ekvivalencia-osztályok definiálhatók.

- A tesztelő a tesztesetek készítésekor elemzi a kódot, hogy biztosítsa minden utasítás legalább egyszeri végrehajtását (az összes lehetséges út-kombináció tesztelésére nincs reális lehetőség).

17.5.1. Útvonal tesztelés

Az útvonal tesztelés strukturális tesztelési stratégia. Célja, hogy minden független útvonalon végighaladjon a teszt. Ekkor legalább egyszer biztosan sor került minden utasítás végrehajtására, és minden feltételes utasítás igaz és hamis eseteire.

- A kiindulás a program folyamatgráfja, amely a döntéseket reprezentáló csomópontokból és a vezérlés irányát képviselő élekből áll. Előállítása viszonylag egyszerű, ha programban nincs goto.
- Csak kisebb programok tesztelhetők ilyen módon.

17.5.2. Ciklomatikus komplexitás

A független utak száma a programban. A CC megmutatja, hogy hány tesztet kell végrehajtani az összes független út végrehajtásához, vagyis minden vezérlő utasítás legalább egyszeri végrehajtásához. Nem lehet a független utak összes kombinációját végrehajtani.

Egy program ciklomatikus komplexitása:

$$CC = lekszma^{\smile} Csompontokszma + 2$$

17.6 Ismertesse az integrációs tesztelési stratégiákat! Mi az összefüggés e stratégiák és a szoftverfolyamat modellje között?

• Teljes rendszerek vagy alrendszerek tesztelése, amelyek előzőleg már tesztelt komponensekből állnak. • A komponensek együttműködéséből származó hibák feltárására szolgál. • Az integrációs teszt fekete doboz tesztelés, a tesztek a specifikációból származnak. • Komplex rendszerben az észlelt hibás eredményből nehéz a hiba helyére következtetni. • Az inkrementális integrációs tesztelés némileg segít

17.6.1. Az integrációs tesztelés stratégiái

Fentről lefelé tesztelés: A rendszer magas szintű komponenseit még a tervezés és az implementáció alatt integrálják. A még el nem készült komponenseket azonos interfésszel készült „csonkok" helyettesítik ahol szükséges. Ezeket fokozatosan kicserélik a kész elemekkel. *(Evolúciós fejlesztésnél alkalmazható)*

Lentről felfelé tesztelés: A hierarchia alsó szintjein lévő modulok integrálásával és tesztelésével kezdik, ahol a magasabb szinteket tesztgenerátorok helyettesítik. *(Inkrementális és újrafelhasználás alapú fejlesztésnél)*

A gyakorlatban a kettő kombinációját alkalmazzák.

17.6.2. A tesztelési stratégiák

Szerkezeti validáció: A fentről lefelé teszteléssel felfedhetők a hibák a rendszerarchitektúrában és a magas szintű tervekben, még a folyamat korai szakaszában. Ez a lentről felfelé tesztelésnél csak később lehetséges.

Rendszerdemonstráció: A fentről lefelé integráció korán lehetővé teszi a korlátozott demonstrációt. Újrafelhasználható komponensek alkalmazásával a lentről felfelé megközelítéssel is lehetséges.

Tesztimplementáció: A programcsonkokat nehéz implementálni, a lentről felfelé tesztelés tesztmeghajtóit valamivel egyszerűbb, de mindenképpen jelentős addicionális fejlesztést igényel.

Tesztmegfigyelés: A tesztek eredményét mindkét módszernél nehéz megfigyelni. Mesterséges környezetre, extra kódra van szükség. Különösen a fentről lefelé megközelítésnél, ahol a magasabb szintek nem szolgáltatnak outputokat.

18 A szoftver minőség fogalma, minőségbiztosítási szabványok, a minőség tervezése, szoftverkarbantartás

18.1 A szoftver minősége

A minőség általában azt jelenti, hogy a termék megfelel specifikációjának. A szoftver esetében ezt nehéz értelmezni, mert *eltérő a megrendelő és a fejlesztő minőségi elvárása*:

- A megrendelő azt várja, hogy a szoftver legyen gazdaságos, megbízható, stb.
- A fejlesztő minőségi követelményei: karbantarthatóság, újrafelhasználhatóság.
- Egyes minőségi kritériumokat nem lehet egyértelműen definiálni (pl. karbantarthatóság, hordozhatóság)
- A szoftver specifikációját nehéz teljessé tenni, tehát a specifikációnak való megfelelés nem garantálja, hogy a felhasználó elégedett lesz a termékkel.

18.2 Minőségkezelés a szoftverprojektben

Minőségbiztosítás: Szabványok és szervezeti eljárások alkalmazása.

Minőségtervezés: Egy konkrét projekthez alkalmas eljárások és szabványok kiválasztása és adaptálása.

Minőségellenőrzés: Annak biztosítása és ellenőrzése, hogy a fejlesztő csapat alkalmazza a minőségi szabványokat és eljárásokat.

A minőségkezelés lehetőleg legyen független a projektvezetéstől

18.2.1. Minőségtervezés

A minőségi tervet a folyamat korai szakaszában kell elkészíteni. A minőségi terv meghatározza a termék minőségi jellemzőit, kijelöli a mérés módját és az alkalmazandó folyamatokat. Meg kell határozni, hogy milyen szervezeti szabványokat kell alkalmazni. Ha szükséges, új szabványokat dolgoznak ki.

A minőségi terv tartalma:

- A termék bemutatása
- Terméktervek
- A folyamatok leírása
- Minőségi célok
- Kockázatok és kockázatkezelés

A minőségi tervnek rövidnek, tömörnek kell lennie (különben nem olvassák el!).

18.2.2. Minőség felülvizsgálat

A minőségi felülvizsgálat elterjedt módszer a folyamatok és a termékek minőségének ellenőrzésére. Egy minőségellenőrzési csoport átnézi a folyamatot, a dokumentációkat és a szoftvert, hogy felfedje a lehetséges hibákat.

A felülvizsgálat típusai:

- A **terv vagy a program vizsgálata**, mint a Verifikáció Validáció esetén (a termék minőségét vizsgálja)
- Az **előrehaladás vizsgálat**a (a folyamat és a termék minőségét vizsgálja)
- A **minőség vizsgálata** (a folyamat és a termék minőségét vizsgálja)

A felülvizsgálat folyamata:

- Szakértők egy csoportja figyelmesen átvizsgálja a szoftver komponenseit, a teljes szoftvert és a dokumentációkat.
- Átnézik a specifikációkat, terveket, kódot, teszttterveket.
- Az eredményes felülvizsgálat a szoftver vagy a dokumentáció elfogadását jelenti. Az észrevételek kijavítása után újabb felülvizsgálatra kerülhet sor.
- A vezetés a felülvizsgálatok eredményei alapján követheti a projekt előrehaladását.

A felülvizsgálat eredményei A felülvizsgálat megállapításait osztályozni kell:

- Nincs tennivaló, a szoftver és a dokumentáció rendben van.
- Javításra visszaadva, a tervezőnek vagy a programozónak ki kell javítania a felfedett hibákat.

- Teljes újragondolás (újratervezés) szükséges. A felfedett hiányosságok a tervek más részeit is érintik.
- A követelmény- és specifikációs hibákról a megrendelőt is értesíteni kell.

18.3 Minőségbiztosítási szabványok

A szabványok adják a keretet a hatékony minőségkezeléshez. Lehetnek: nemzetközi-, nemzeti-, szervezeti- és projektszabványok.

A **termékszabványok** olyan tulajdonságokat írnak elő, amelyek a termék minden elemére nézve kötelezőek:

- Dokumentációs szabványok (pl. dokumentumok szerkezete): A dokumentációk a szoftver megfogható meg-nyilvánulásai, általuk válik követhetővé a szoftverfolyamat, ezért a dokumentációs szabványok különösen fontosak. Típusai:
  - A dokumentálás folyamatának szabványai: Hogyan kell a dokumentumokat elkészíteni, validálni, karbantartani.
  - Dokumentumszabványok: A dokumentumok tartalma, szerkezete, megjelenése.
  - A dokumentumcsere szabványai: A dokumentumok tárolása, különböző dokumentációs rendszerek közti cseréje.
- Kódolási szabványok (programozási stílus, programnyelv használat)

A **folyamatszabványok** a szoftverfejlesztés alatt követendő folyamatokat határozzák meg (pl. a specifikáció, tervezés, stb. folyamata, módszerei, dokumentumai).

A szabványok a legjobb gyakorlat és a korábbi projektek hibáinak összegyűjtött adatai alapján készülnek. Nemzeti és nemzetközi szervezetek (IEEE, ANSI, BSI, NATO, stb.) dolgozzák ki különböző projektekre.

Kiterjednek a szoftvertervezés terminológiáira, programozási nyelvekre, jelölésrendszerre, programozási módszerekre, ellenőrzésre, validálásra. Folytonosságot biztosítanak egy változó szervezetben, az új résztvevők a helyi szabványok megismerésével hamarabb be tudnak kapcsolódni a munkába.

18.4 Mi az összefüggés a szoftverfolyamatok és az előállított szoftver minősége között?

Egy hagyományos termék minősége alapvetően függ az előállítása során alkalmazott folyamatok minőségétől (pl. iparszerű gyártásnál). Az ipari gyártásban egyértelmű az összefüggés a termék minősége és az előállítási folyamat minősége között. Ez a szoftverfejlesztésnél is így van, de sok minőségi jellemző nehezen mérhető, számszerűsíthető, illetve szoftverfejlesztésben az egyéni képzettség és gyakorlat különösen fontos. Emellett a szoftvert egyedileg tervezik, a szoftverfejlesztés nem mechanikus folyamat.

A szoftverfejlesztés folyamata és a termék minősége között erős összefüggés van, de ez nagyon összetett és alig megfogható. Külső tényezők, mint az alkalmazás újszerűsége, vagy a piacra vitel siettetése is befolyásolják a minőséget. Figyelembe kel venni, hogy alkalmazható-e az adott projektre egy szabványos folyamat.

18.5 Az ISO 9000 szabvány

A minőségkezelés nemzetközi szabvány-rendszere (ISO – International Standardization Organisation). Sokféle szervezetre alkalmazható, a termeléstől a szolgáltatásokig. A megújított ISO 9000:2000 szabvány már foglalkozik a felhasználói elégedettség mérésével is.

Az ISO 9001 alkalmazható a tervezéssel, fejlesztéssel, karbantartással foglalkozó szervezetekre. Az ISO 9001 egy általános minőségkezelési folyamat, amelyet adaptálni kell a konkrét szervezetre.

A vonatkozó minőségi szabványokat és eljárásokat a szervezet minőségi kézikönyvében kell lefektetni. Egy független, külső bizottság tanúsítja, hogy a szervezet minőségi kézikönyve és gyakorlata megfelel az ISO 9000 szabványnak. A tanúsítást évente felülvizsgálják és megújítják, vagy megvonják a szervezettől. A megrendelők (pl. közbeszerzésben) mind gyakrabban írják elő feltételként az ISO 9000 tanúsítványt.