Szoftvertechológia alapjai

Vizsgakérdések

Dallos Ádám, Majnár László

2016. május 24.

Tartalomjegyzék

1. Mi a szoftver? Sorolja fel azokat a termékeket, amelyek a szoftverhez tartoznak.

A szoftver: Számítógép-programok és a hozzájuk tartozó dokumentációk összessége (Somerville def.) A gyakorlatban hozzátartoznak a szakterületi ismeretek és azok dokumentációi is, amelyek alapján a szoftvert kifejlesztették.
A szoftverhez tartozó termék: A programok és a hozzá kapcsolódó dokumentációk.

2. Mi a szoftverfolyamat? Sorolja fel a szoftverfolyamat főbb tevékenységeit.

A szoftverfolyamat: A szoftver termék előállítására irányuló tevékenységek sora.
Főbb tevékenységek:

1. **Szoftverspecifikáció:** A szoftver feladatainak és a megszorításoknak specifikációja.
2. **Szoftverfejlesztés:** A szoftver rendszer tervezése és elkészítése (programkészítés, tesztelés).
3. **Szoftvervalidáció:** Annak bizonyítása, hogy az előkészített rendszer a felhasználó elvárásainak megfelelően működik.
4. **Szoftverevolúció:** A szoftver karbantartása és továbbfejlesztése a változó igényeknek megfelelően.

3. Sorolja fel a szoftverfolyamat általános modelljeit és jellemezze azokat néhány szóban.

A szoftverfolyamat modellje a folyamat absztrakt reprezentációja.

- **Vízesés modell:** Az alapvető tevékenységek önálló fázisok a folyamat során.
- **Evolúciós fejlesztés:** A specifikáció, a fejlesztés és a validáció összefonódik. Ez az alapja a manapság elterjedt alkalmazott agilis fejlesztési módszereknek.
- **Formális transzformációk:** A követelmények matematikai modelljéből, formális transzformációval állítja elő a szoftvert. Ez a módszer az őse a mai modellközpontú fejlesztésnek.
- **Integráció újrafelhasználható komponensekből:** A rendszert meglévő komponensek integrálásával állítja elő.

4. Miért van szükség arra, hogy a szoftvertervezők számára etikai kódexet állítsanak össze? Sorolja fel a fontosabb etikai előírásokat.

Szükségesség:

- **Bizalmasság:** A szoftvertervezőnek tisztelnie kell a munkaadója, vagy megrendelője bizalmát, attól függetlenül, hogy aláírtak-e titoktartási nyilatkozatot.
- **Hozzáértés:** A tervező nem vállalhat el olyan munkát, amely meghaladja képességeit, nem tűntetheti fel hamis színben tudását, képességeit.
- **Szellemi jogok:** Tiszteletben kell tartania a szellemi tulajdonjogokat.
- **Számítógépes visszaélések:** A tervező nem használhatja fel ismereteit arra, hogy másoknak közvetve, vagy közvetlenül kárt okozzon.

Előírások:

- A helyi, nemzeti és nemzetközi szabályok ismerete és betartása.
- Etikus és bizalmat keltő viselkedés, ami több, mint pusztán a törvények betartása.
- A megbízó titkainak megtartása.
- A szellemi tulajdonjogok tiszteletben tartása.
- A számítógépes visszaélések megakadályozása, ill. elkerülése.

5. Melyek az eredendő rendszertulajdonságok? Hogyan csoportosítjuk őket?

A rendszer több, mint komponenseinek halmaza. Alapvető tulajdonságai többnyire nem származtathatóak a komponensek tulajdonságaiból. A rendszer eredendő tulajdonságai a komponensek közti összetett kapcsolatok, kölcsönhatások következményei. Ezért az eredendő tulajdonságok csak akkor állapíthatók meg és válnak mérhetővé, mikor megtörtént a komponensek integrációja.
Eredendő rendszertulajdonságok:

- **A rendszer súlya:** Ez kiszámítható a komponensek súlyából.
- **A rendszer megbízhatósága:** Nemcsak a rendszer komponenseitől, hanem azok kapcsolatától is függ.
- **A rendszer használhatósága:** Összetett tulajdonság, amely nemcsak a hardver és szoftver komponensektől függ, hanem a környezettől és a működtető, sőt a felhasználó emberektől is.

Tulajdonságok típusai:

- **Funkcionális tulajdonságok:** A funkcionális tulajdonságok akkor figyelhetők meg, ha a rendszer minden része együttműködik egy cél elérése érdekében. (pl. kerékpár!)

- **Nem funkcionális tulajdonságok:**
  - Olyan rendszertulajdonságok, amelyeket a rendszer működési környezete is befolyásol. Ezek gyakran kritikusak a számítógép alapú rendszerek működése szempontjából, ezért ezeket már a tervezéskor definiálni kell.
  - Ilyenek lehetnek a megbízhatóság, a teljesítmény, a biztonságosság, vagy a védelem.

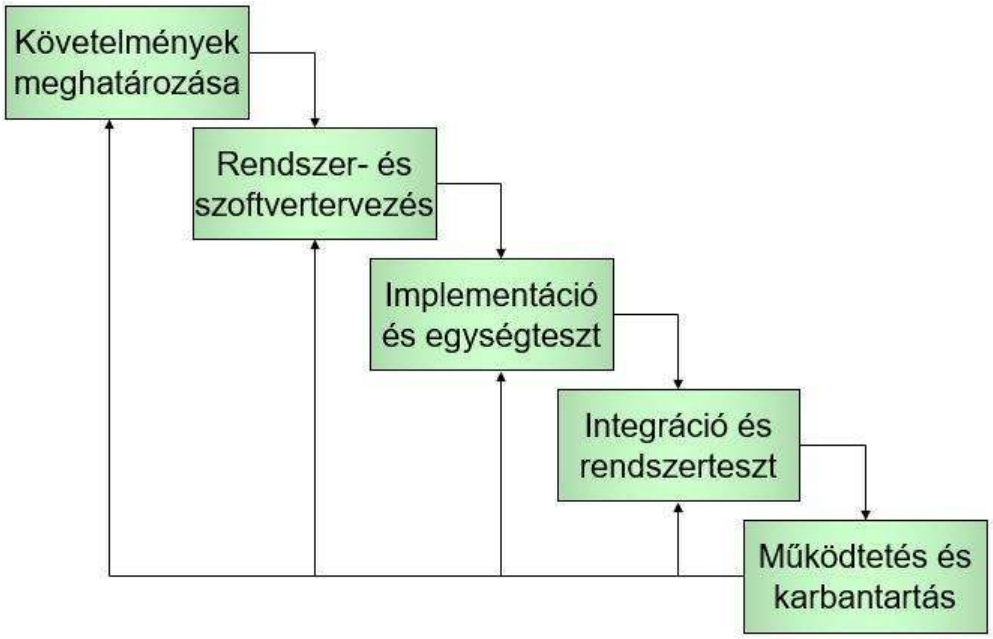
6. Mi a különbség a funkcionális és a nem-funkcionális rendszertulajdonságok között?

A **funkcionális tulajdonságok** akkor figyelhetők meg, ha a rendszer minden része együttműködik egy cél érdekében.
A **nem funkcionális tulajdonságok** pedig olyan rendszertulajdonságok, amelyeket a rendszer működési környezete is befolyásol. Pl.: teljesítmény, megbízhatóság, biztonság, védelem.

7. Melyek azok a tevékenységek, amelyek közösek minden szoftverfolyamatban?

- **Szoftverspecifikáció:** A szoftver feladatainak és a megszorításoknak specifikációja.
- **Szoftverfejlesztés:** A szoftver rendszer tervezése és elkészítése (programkészítés, tesztelés).
- **Szoftvervalidáció:** Annak bizonyítása, hogy az előkészített rendszer a felhasználó elvárásainak megfelelően működik.
- **Szoftverevolúció:** A szoftver karbantartása és továbbfejlesztése a változó igényeknek megfelelően.

8. Vázolja fel a vízesés modellt, sorolja fel a modell előnyeit és hátrányait!



A vízesés modell előnyei:

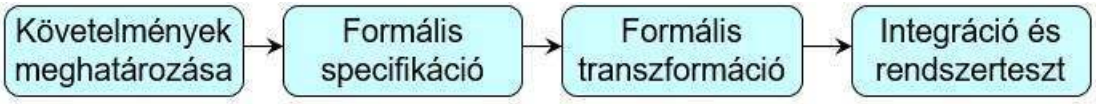
- Jól áttekinthető és követhető fejlesztési projekt folyamatot eredményez.
- A folyamat termékei szerződésekkel könnyen lefedhetők. (specifikációs és tervezési dokumentumok, program(ok), stb.)
- A tevékenységek jól, és pontosan tervezhetők.

A vízesés modell hátrányai:

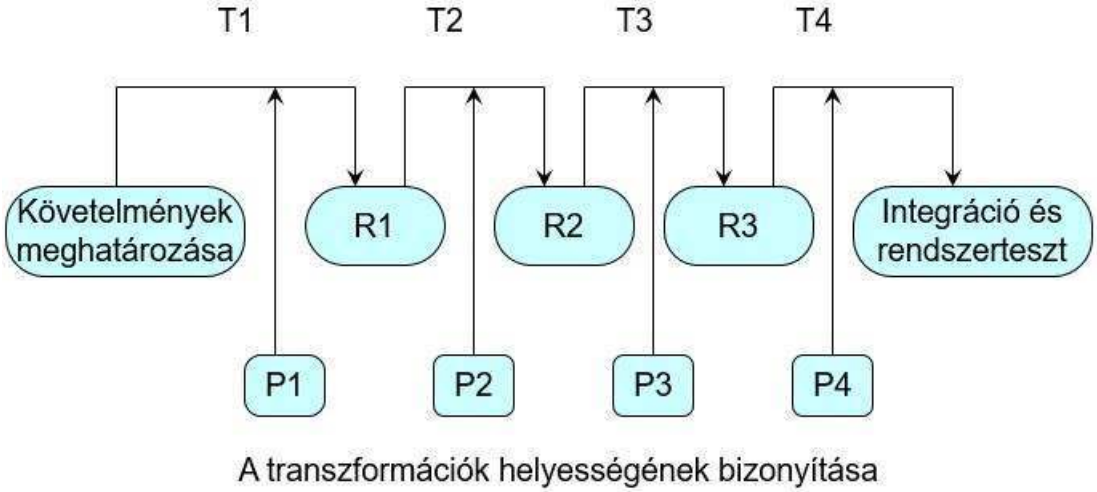
- Egymástól elkülönült fázisokra osztja a projektet (költséges egy korábbi fázishoz visszatérni pl. specifikációs, vagy tervezési hiba esetén).
- Csak a projekt végén, az átadáskor (a működtetés első lépésekor) derülnek ki a specifikációs hibák.
- Nem képes rugalmasan alkalmazkodni a felhasználói igények változásaihoz.
- Csak a követelmények pontos ismeretében alkalmazható.

9. Mi a formális rendszerfejlesztés? Milyen előnyei és hátrányai vannak?

A vízesés modellhez hasonló, de a fejlesztés formális matematikai eszközökkel állítja elő a futtatható programot a rendszerspecifikáció matematikai modelljéből, több transzformációs lépésen keresztül.
Minden transzformáció során, lépésenként kell végrehajtani a tesztelést.



Formális transzformációk:



A transzformációk helyességének bizonyítása

Formális rendszerfejlesztés előnyei:

- Kritikus rendszerek esetén, ahol kulcskérdés a biztonságosság, megbízhatóság, vagy védelem.
- A transzformáció és a bizonyítás részben automatizálható.

Formális rendszerfejlesztés hátrányai:

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

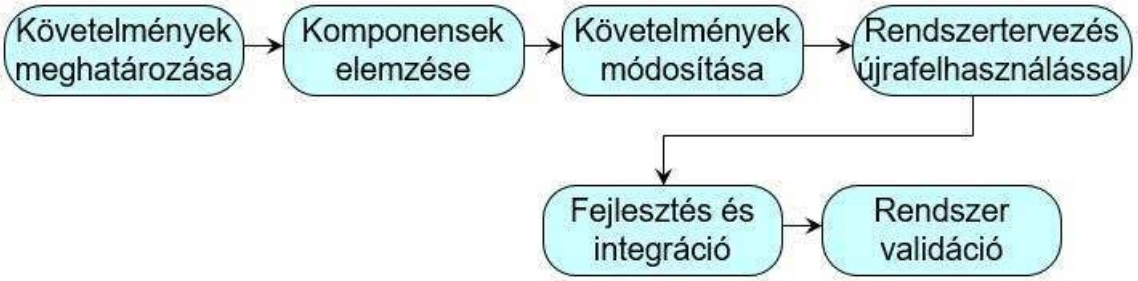
10. Mi az evolúciós fejlesztési modell lényege? Miért nehéz karbantartani az így fejlesztett programokat?

Az alapgondolat: Ki kell dolgozni egy kezdeti implementációt, amelyet a felhasználó véleményezhet, és azt kell finomítani az elfogadásig. A specifikáció a fejlesztés és validáció párhuzamosan folytatható tevékenységek.
Feltáró fejlesztés: A követelmények feltárása lépésenként, a megrendelővel együttműködve történik, folyamatosan kiegészítve a rendszert az új funkciókkal, részekkel.(Ilyen az Agile módszerek többsége.)
Eldobható prototípus: „Deszkamodellek” készítése és átadása az ügyfélnek, a követelmények pontosabb feltárása érdekében.
Miért nehéz karbantartani? Azért nehéz karbantartani, mert minden művelet egyszerre folyik, és nincs részletes specifikáció.

11. Mi az újrafelhasználás orientált fejlesztés lényege? Vázolja fel a folyamatot! Milyen esetekben alkalmazható?

Már létező, újrafelhasználható szoftver-komponensek egységes szerkezetbe való integrálása (komponens alapú rendszerfejlesztés). Polcról levehető termékeket használ fel a fejlesztés során. Gyakran beépül a korábban ismertetett folyamatokba. Az elérhető komponenseket megtalálni, azokat integrálni nem egyszerű.

A folyamat modell:



Milyen esetekben alkalmazható?

Minőségjavítása, költségek és a fejlesztési idő csökkentésének érdekében használható. Csak akkor használjuk, ha tudjuk, hogy az adott komponens megfelelően működik. Nagy rendszerek építésekor gyakran használt stratégia a COTS termékek integrálása. Különösen a gyors fejlesztést kívánó eCommerce, eBusiness rendszerek körében. A fejlesztési idő nagyságrendekkel csökkenthető.

12. Mi a különbség a CASE eszközök, -eszközkészletek és -környezetek között?

- **Eszközök (tools):** Az egyes folyamatlépéseket támogatják, mint a terv konziszetciájának ellenőrzése, program fordítás, teszteredmények összehasonlítása, stb.
- **Eszközkészletek (workbench):** A szoftverfolyamat egyes fázisait támogatják, mint pl. specifikáció, vagy tervezés. Általában több, egymással együttműködő eszközből állnak.
- **Környezetek (environments):** A szoftverfolyamat több fontos, vagy valamennyi részét támogatják. Legtöbbször több, integrált eszközkészletből állnak.

13. Miért célszerű projektszervezetben végezni a szoftverfejlesztést?

Azért célszerű, hogy a szoftver a tervezett ütemezés szerint, határidőre, a követelményeknek megfelelően készüljön el. Továbbá a projekt menedzselésére azért van szükség, mert a szoftverfejlesztés mindig kötött pénzügyi és megszabott időkeretek között folyik, amelyeket a megrendelő, vagy a fejlesztő szervezet jelöl ki.

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

Mivel a szoftver nem kézzel fogható, továbbá a szoftverfejlesztés folyamata sincs szabványosítva, így sokkal nehezebb meghatározni, hogy mennyi időbe telik a fejlesztés, mennyi ember kellhet hozzá, mekkora ennek a költsége. Továbbá a megrendelő számára nem nagyon tud a projekt elkészültéig kézzel fogható programot mutatni, amely igen csak megnehezíti a dolgokat.

- A szoftverfejlesztési projekt gyakran egyedi, amely nem általánosítható folyamat.
- A projekt tervezést alfeladatokra kell bontani, és ütemezni.
- Némely feladat futhat párhuzamosan is, ha a feladatok nem függenek egymástól.

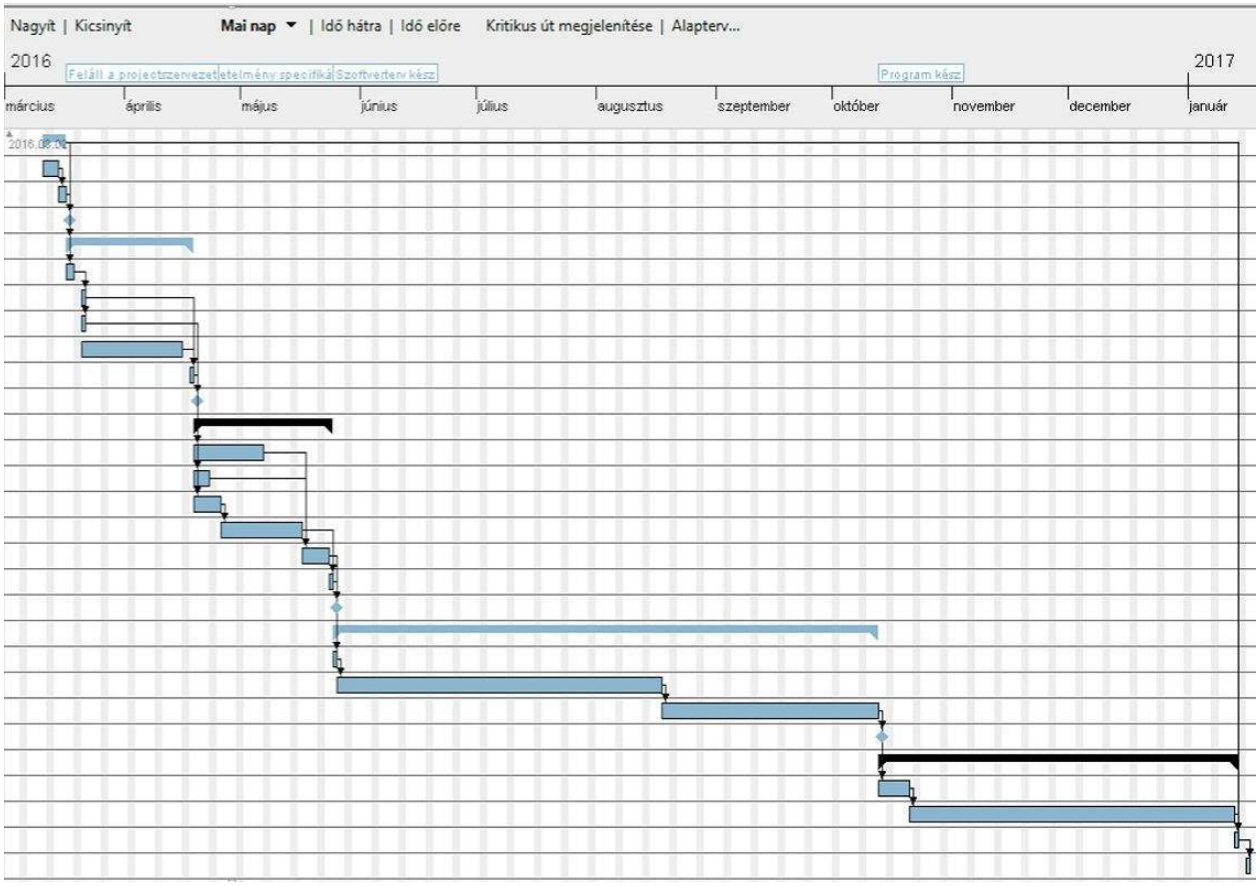
15. Milyen típusú terveket kell készíteni egy projekt tervezésekor?

- **Munkaterv, ütemezés:** Az elvégzendő feladatok, ütemezésük és összefüggéseik tervezése.
- **Minőségi terv:** Meghatározza a projektben használandó minőségbiztosítási eljárásokat és szabványokat.
- **Validációs terv:** Meghatározza a rendszer validációja során használandó módszereket, erőforrásokat, ütemezést.
- **Konfiguráció-kezelési terv:** Leírja a konfiguráció-kezelés eljárásait és struktúráját.
- **Karbantartási terv:** A karbantartás követelményeinek, költségeinek terve.
- **Munkaerő-fejlesztési terv:** Terv a projekten dolgozó csapat szaktudásának, tapasztalatainak fejlesztésére.

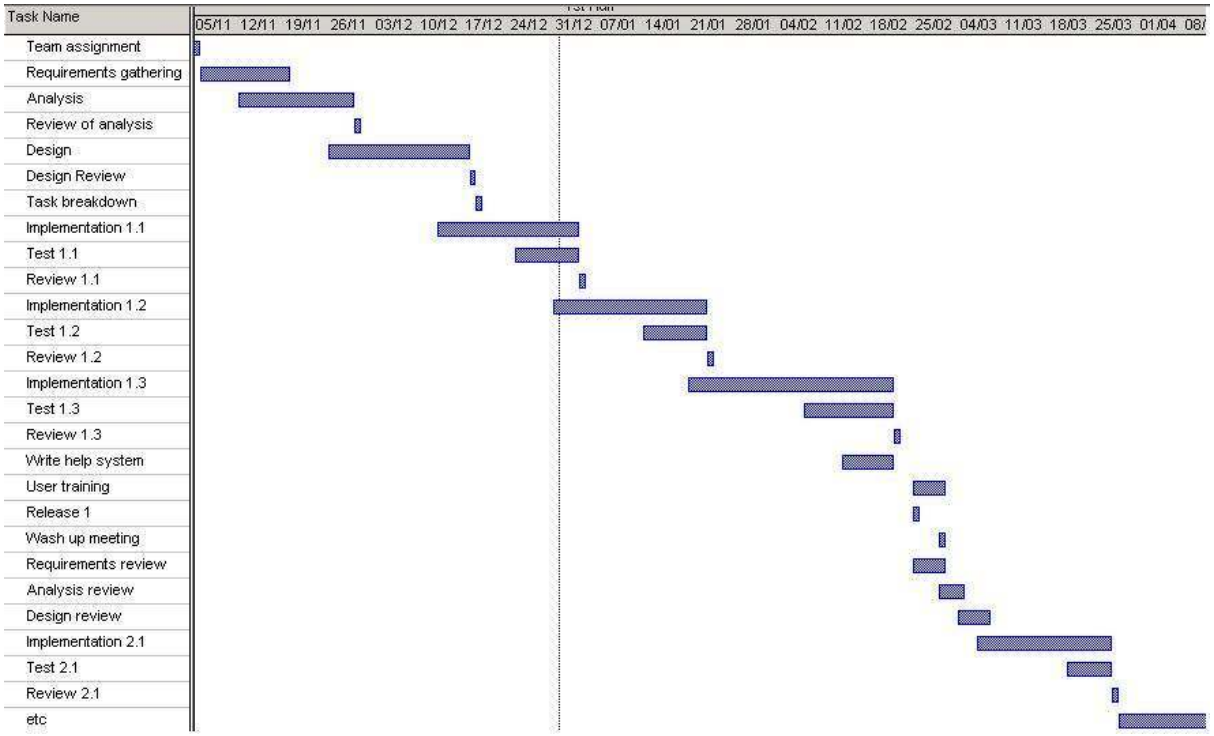
16. Vázolja fel egy vízesés modell szerint végzendő fejlesztési projekt ütemezését oszlopdiagram formában!

SoWa házifeladatból:

Név	Kezdő dátum	Záró dátum
Projektindítás	2016.03.11.	2016.03.16.
Projektszervezet felállítása	2016.03.11.	2016.03.14.
Piac elemzése	2016.03.15.	2016.03.16.
Feláll a projectszervezet	2016.03.17.	2016.03.17.
Követelmények meghatározása	2016.03.17.	2016.04.18.
Megvalósíthatósági tanulmány	2016.03.17.	2016.03.18.
Erőforrásigények felmérése	2016.03.21.	2016.03.21.
Követelmény specifikáció	2016.03.21.	2016.03.21.
Demoprogram írása	2016.03.21.	2016.04.15.
Áruházzal való szerződés	2016.04.18.	2016.04.18.
Követelmény specifikáció kész	2016.04.19.	2016.04.19.
Szoftvertervezés	2016.04.19.	2016.05.24.
Adatgyűjés	2016.04.19.	2016.05.06.
VPS bérlése	2016.04.19.	2016.04.22.
Algoritmusok kiválasztása	2016.04.19.	2016.04.25.
Szoftvermodell készítése	2016.04.26.	2016.05.16.
Adatok felvétele az adatbázisba	2016.05.17.	2016.05.23.
A tervek validálása	2016.05.24.	2016.05.24.
Szoftverterv kész	2016.05.25.	2016.05.25.
Szoftver implementáció	2016.05.25.	2016.10.12.
Programtervezés	2016.05.25.	2016.05.25.
Programozás	2016.05.26.	2016.08.17.
Program tesztelés	2016.08.18.	2016.10.12.
Program kész	2016.10.13.	2016.10.13.
Integráció, rendszertervezés	2016.10.13.	2017.01.13.
Modulok intergrálása	2016.10.13.	2016.10.20.
Rendszerteszt	2016.10.21.	2017.01.12.
Rendszerteszt kész	2017.01.13.	2017.01.13.
Átadás	2017.01.16.	2017.01.16.


Nagyít | Kicsinyít
Mai nap
| Idő hátra | Idő előre
Kritikus út megjelenítése | Alapterv...
2016
2017
Feláll a projektszervezet
Ietelmény specifikál
Szoftverterv kész
Program kész
március
április
május
június
július
augusztus
szeptember
október
november
december
január

17. Kísérelje meg felvázolni egy evolúciós folyamat szerint végzendő fejlesztési projektterv oszlopdiagramját!



18. Miért iteratív tevékenység a szoftverprojekt tervezése?

A szoftverprojekt tervezése egy folyamatos tevékenység. Amely a koncepció kidolgozásától a rendszer átadásáig tart. Az előrehaladást folyamatosan követni kell, projekttervet rendszeresen felül kell vizsgálni és át kell dolgozni a változó állapotnak megfelelően (ezért iteratív).

19. Milyen kockázatokat különböztethetünk meg egy szoftverfejlesztési projektben? Sorolja fel és jellemezze őket!

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

- **Projekt kockázat:** A projekt ütemtervét, vagy az erőforrásokat veszélyezteti.
- **Termék kockázat:** A szoftver minőségét, vagy teljesítményét veszélyezteti.
- **Üzleti kockázat:** A szoftver beszerzését, értékesítését veszélyezteti.
- **Szervezeti kockázat:** A fejlesztést végző szervezetet veszélyezteti.

20. Sorolja fel a fontosabb szerepeket egy projektszervezetben! Ismertesse néhány szóban az egyes szerepek tevékenységeit!

- **Projektvezető (szakmai - adminisztratív vezető):**
  A projektvezető felelős a projektcélok megvalósulásáért. Szakmai és adminisztratív vezető, aki felelős a projekt sikeres, határidőre való befejezéséért, a kezdetben meghatározott keretek között.
- **Rendszertervező:**
  A rendszertervezés a számítógép alapú rendszerek tervezésével foglalkozik (hardver, szoftver, folyamatok) a szoftvertervezés ennek egy része.
- **Vezető programozó:**
  A programozási folyamat előrehaladásáért felelős személy. Körülötte helyezkednek el az alábbiak.
- **Tartalék programozó:**
  Vezető programozó mellett lévő gyakorlott programozó.
- **Könyvtáros (adminisztrátor):**
  A fejlesztési projekt dokumentációinak és termékeinek (verziók) elkészítéséért (készíttetéséért), rendben tartásáért felelős.
- **Tesztelő:**
  A modellek és termékek tesztelését, verifikálását végzi.
- **Minőségfelelős:**
  A minőségbiztosítási szabályok betartásáért felel a teljes szoftverfolyamat során.
- **Dokumentátor**
- **Operációs rendszer szakértő:**
  Az operációs rendszer kiválasztásáért felelős személy.
- **Adatbázis szakértő**
- **Szakterületi specialista:**
  Az alkalmazási terület ismerője, aki már részt vett hasonló rendszer kidolgozásában.
- **CASE eszköz szakértő:**
  A fejlesztés során felhasználható CASE eszközök kiválasztásáért felelős személy.

21. Mi a feladata a megvalósíthatósági tanulmánynak. Hol van a helye a szoftverfolyamatban?

Feladata: Megalapozni azt a döntést, hogy érdemes-e a tervezett rendszert megvalósítani.
Az alábbi kérdésekre ad választ:

- Más, hasonló szervezeteknél milyen megoldásokat alkalmaztak hasonló feladatokra.
- Mennyiben támogatja a rendszer a megrendelő általános célkitűzéseit.
- Megvalósítható-e a rendszer a tervezett költségen belül, az adott technológiával, a kívánt határidőre.
- Integrálható-e a rendszer más, már meglévő rendszerekkel.

Helye a szoftverfolyamatban: A megvalósíthatósági tanulmány helye a **követelménytervezés folyamatában** van a szoftverfolyamat során.

22. Sorolja fel a legfontosabb szempontokat, amelyeket egy tervezőnek a felhasználói követelmények specifikálásakor ügyelnie kell!

- A felhasználói követelményeket úgy kell megfogalmazni, hogy az informatikában járatlan felhasználó is megértse.
- Ezért itt nem célszerű modelleket alkalmazni, hanem természetes nyelven, táblázatokkal és diagramokkal kell a felhasználói követelményeket érthetővé és egyértelművé tenni.
- A természetes nyelv alkalmazásának nehézségei:
  - Az egyértelműség és pontosság hiánya
  - A követelmények keveredése
  - A követelmények ötvöződése
- Dolgozzunk ki egységes formátumot az összes követelmény leírására.
- Használjuk a nyelvet következetesen, pl. a szükséges követelményeket a „kell”, a kívánatos követelményeket pedig a „javallott” szóval jelölhetjük.
- Készítsünk glosszáriumot a szövegben használt fogalmak és rövidítések magyarázatára.
- A fontos részeket vizuálisan is emeljük ki a szövegből.
- Kerüljük a számítógépes zsargon használatát.

23. Milyen veszélyei vannak a természetes nyelv használatának a követelmények specifikálásakor? Milyen módon lehet ezeket csökkenteni?

- **Félreérthetőség:** A szöveg írója és olvasója egy adott kifejezésen más fogalmat ért, vagy azonos fogalmakat eltérő módon értelmez.
- **Félreérthetőség csökkentése:** Egy adott fogalom, definíció minél pontosabb, érthetőbb leírása az író által. Akár példák segítségével való megfogalmazás.
- **Túlzott rugalmasság:** Ugyanaz a követelmény sokféle módon írható le.
- **A modularitás hiánya:** A természetes nyelvben nincs egyértelmű lehetőség az összefüggések jelölésére. Így egy követelmény megváltozásakor az összes követelményt át kell vizsgálni, hogy a kapcsolódó követelményeket megtaláljuk.
- **Strukturált nyelvű specifikáció alkalmazása:**
  - A természetes nyelv szabályozott, strukturált alkalmazása a követelmények leírására.
  - Egyértelműbbé teszi a követelmények specifikációját, de közben megtartja a természetes nyelv rugalmasságát.
  - A struktúrát űrlapok vagy sablonok alkalmazásával támogatják. Kötelező információk feltüntetésével az űrlapon.

24. Egy nagy rendszer fejlesztése során kiknek kell részt venniük a felhasználói követelmények verifikálásában? Miért?

25. Sorolja fel a rendszermodellek típusait és jellemezze azokat egy mondatban!

- **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.

26. Melyek a legfontosabb különbségek a felhasználói és a rendszerkövetelmények specifikálása között? Kiknek szólnak az egyes specifikációk?

Felhasználói követelmények: A rendszer szolgáltatásainak közérthető leírása, diagramokkal, táblázatokkal, ábrákkal. Itt nem célszerű modelleket alkalmazni, az egyértelműség érdekében.
Rendszerkövetelmények:

- Strukturált dokumentum a rendszer szolgáltatásainak részletes leírásával (funkcionális specifikáció). Ez a szerződés alapja.
- A rendszerkövetelmények a felhasználói követelmények részletesebb és rendezett leírását adják.
- A rendszertervezés alapjául szolgálnak, tartalmazhatják a rendszer modelljeit.
- A rendszerkövetelmények leírják, hogy a rendszernek mit kell elvégeznie, majd a tervek határozzák meg, hogy hogyan tegye.

Kinek a számára készülnek az egyes dokumentumok?

- **Felhasználói követelmények specifikációja:** Megrendelő vezetőségének, a rendszer végfelhasználóinak, rendszertervezőknek és a szerződéskötőknek.
- **Rendszer követelmények specifikációja:** A rendszer végfelhasználóinak, a megrendelő középvezetőinek, a rendszertervezőknek és szoftverfejlesztőknek.
- **Szoftver specifikáció:**A rendszertervezőknek, szoftverfejlesztőknek, tesztelőknek és karbantartóknak.

27. Melyek a prototípuskészítés céljai? Milyen prototípusok léteznek, melyik milyen célból készül?

Céljai:

- A prototípus elsődleges célja az, hogy segítse a felhasználókat a rendszerkövetelmények megértésében.
- A követelménytervezés része, a követelmények feltárásának és validációjának eszköze.
- A szoftverrendszer kezdeti verziója, amely alkalmas a rendszer koncepciójának bemutatására és kipróbálására.
- A prototípus csökkenti a követelményekkel kapcsolatos kockázatokat.

Prototípusok típusai:

- **Evolúciós prototípus:** Célja egy működő rendszer átadása a megrendelőnek. A legfontosabb követelmények implementálásával egyszerű rendszer készül, amelyet újabb követelmények feltárásával fokozatosan egészítenek ki új funkciókkal.
- **Eldobható prototípus:** Célja a követelményspecifikációból fakadó kockázat csökkentése. A prototípust egy kezdeti specifikáció alapján készítik, validálásra átadják a felhasználónak, majd eldobják.

28. Mi a különbség az evolúciós és az eldobható prototípus között? Melyiket mikor érdemes alkalmazni?

- **Evolúciós prototípus:** Célja egy működő rendszer átadása a megrendelőnek. A legfontosabb követelmények implementálásával egyszerű rendszer készül, amelyet újabb követelmények feltárásával fokozatosan egészítenek ki új funkciókkal.
- **Eldobható prototípus:** Célja a követelményspecifikációból fakadó kockázat csökkentése. A prototípust egy kezdeti specifikáció alapján készítik, validálásra átadják a felhasználónak, majd eldobják.
  Az eldobható prototípus nem tekinthető végleges rendszernek, mert:
  - A kivételek, hibák kezelése általában hiányzik.
  - Több rendszertulajdonság kimaradhat a prototípusból.
  - Nem készül specifikáció a hosszú távú karbantartásra.
  - A prototípus még nem a megfelelő struktúra szerint épül.

Alkalmazási területek:

- **Evolúciós prototípus:** Weblap fejlesztésben és e-business alkalmazásokban használják.
- **Eldobható prototípus:** A követelmények validálása: a prototípus felfedheti a félreértéseket, hibákat és hiányosságokat a követelményekben

29. Mit jelent az adatbázis-programozás? Milyen rendszerek fejlesztésére alkalmas?

- Az evolúciós fejlesztés az adatbázison alapuló üzleti alkalmazások területén általánosan alkalmazott technika.
- A kereskedelmi adatbázis-kezelő rendszerek olyan 4GL fejlesztő eszközöket tartalmaznak, amelyek támogatják a lekérdezést/adatkezelést (SQL), táblázatkezelést, jelentés generálást, felhasználói felületek tervezését, stb.
- Az adatfeldolgozási alkalmazásokban sok közös feladat 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.

Adatbázison alapuló rendszerek fejlesztésére alkalmas. (??)

30. Melyek a prototípus készítésének előnyei? Ismertese a prototípuskészítő technikákat! Melyiket milyen esetben célszerű alkalmazni?

Prototípuskészítés előnyei:

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

Prototípuskészítő technikák:

- Evolúciós prototípus készítése:
  - Célja egy működő rendszer átadása a megrendelőnek. (esetleg korlátozott funkcionalitással)
  - A legfontosabb követelmények implementálásával egyszerű rendszer készül, amelyet újabb követelmények feltárásával fokozatosan egészítenek ki új funkciókkal.
  - Az AGILIS fejlesztés alapvető módszere.
  - Weblap fejlesztésben és e-business alkalmazásokban használják.

  - Olyan rendszereknél célszerű alkalmazni, ahol nem készíthető el előre a végleges specifikáció. Ilyenek általában az intenzív felhasználói interfész-használatot igénylő rendszerek.
  - Nincs részletes rendszerspecifikáció, sokszor a részletes követelménydokumentum is hiányzik.
  - A felhasználó már a rendszer fejlesztése közben jelezheti, hogy milyen irányban kívánja folytatni a fejlesztést.
  - A fejlesztéshez gyors, iterálható fejlesztő eszközökre és módszerekre van szükség.
  - Mivel nem készül követelményspecifikáció, a validáció is csak a rendszer bemutatásával történhet.
  - 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, folyamatmodellező nyelvek: BPML-Business Process Modeling Language).
  - A felhasználói felületek GUI fejlesztő eszközökkel készíthetők.

- Eldobható prototípus készítése:
  - Célja a követelményspecifikációból fakadó kockázat csökkentése.
  - A prototípust egy kezdeti specifikáció alapján készítik, validálásra átadják a felhasználónak,majd eldobják.
  - A követelményspecifikáció elkészülte után nem használható fel.
  - Az eldobható prototípus nem tekinthető végleges rendszernek, mert:
    * A kivételek, hibák kezelése általában hiányzik.
    * Több rendszertulajdonság kimaradhat a prototípusból.
    * Nem készül specifikáció a hosszú távú karbantartásra.
    * A prototípus még nem a megfelelő struktúra szerint épül.

31. Mit tenne, ha egy eldobható prototípust a megrendelő meg akarna vásárolni? Milyen érveket hozna fel álláspontja indoklására?

Az eldobható prototípus nem tekinthető végleges rendszernek, mert:

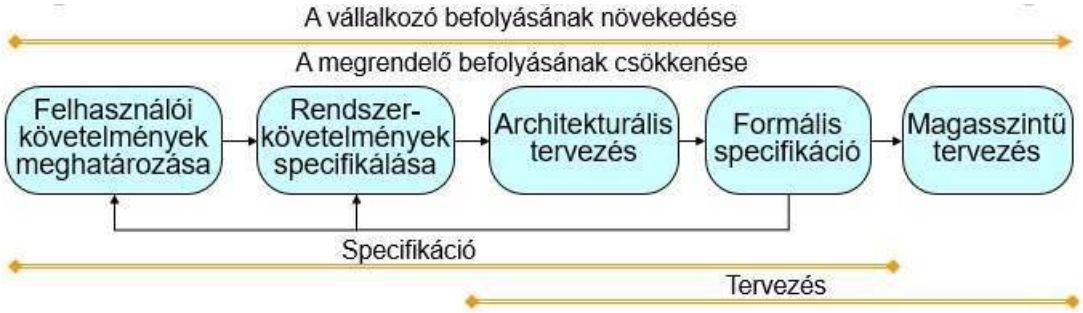
- A kivételek, hibák kezelése általában hiányzik.
- Több rendszertulajdonság kimaradhat a prototípusból.
- Nem készül specifikáció a hosszú távú karbantartásra.
- A prototípus még nem a megfelelő struktúra szerint épül.

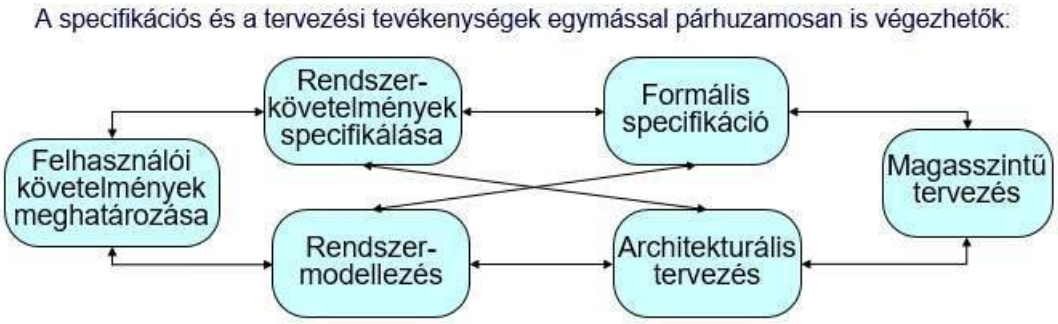
A vezetők gyakran nyomást gyakorolnak a fejlesztőkre, hogy egy működő eldobható prototípust végleges rendszerként adjanak át. Ez nagyon veszélyes, mert:

- Az eldobható prototípus nem alakítható úgy, hogy a nem-funkcionális követelményeknek (teljesítmény, megbízhatóság, skálázhatóság, 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)

32. Ismertesse a formális specifikáció helyét és jelentőségét a szoftverfolyamatban!

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 műveletei 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.

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.

33. Sorolja fel a formális specifikáció előnyeit és hátrányait! Milyen típusú rendszerek specifikálásakor alkalmazzák a formális módszereket?

Formális specifikáció hátrányai: A formális módszerek - az előzetes várakozások ellenére nem tudtak teret hódítani, mert:

- Kialakultak többé-kevésbé sikeres módszerek, mint az OO tervezés, konfigurációkezelés, a strukturált programozás, stb., amelyek javították a szoftver minőségét.
- Újabban a szoftver gyors piacra kerülése fontosabb, mint a minőség. A gyors fejlesztési technikák nem illeszkednek a formális módszerekhez.
- A formális módszerek csak korlátozottan alkalmazhatók például a felhasználói interfészek, felületek és munkafolyamatok specifikálására.
- A formális módszerek nehezen, vagy egyáltalán nem alkalmazhatók nagy rendszerek esetén.
- Még kevés eszköz készült a formális módszerek támogatására.
- A formális módszereknek csak korlátozott felhasználási lehetőségei vannak. Alkalmazásuk kockázata és költsége sok esetben nagyobb, mint a várható előnyök.

A formális specifikáció előnyei:

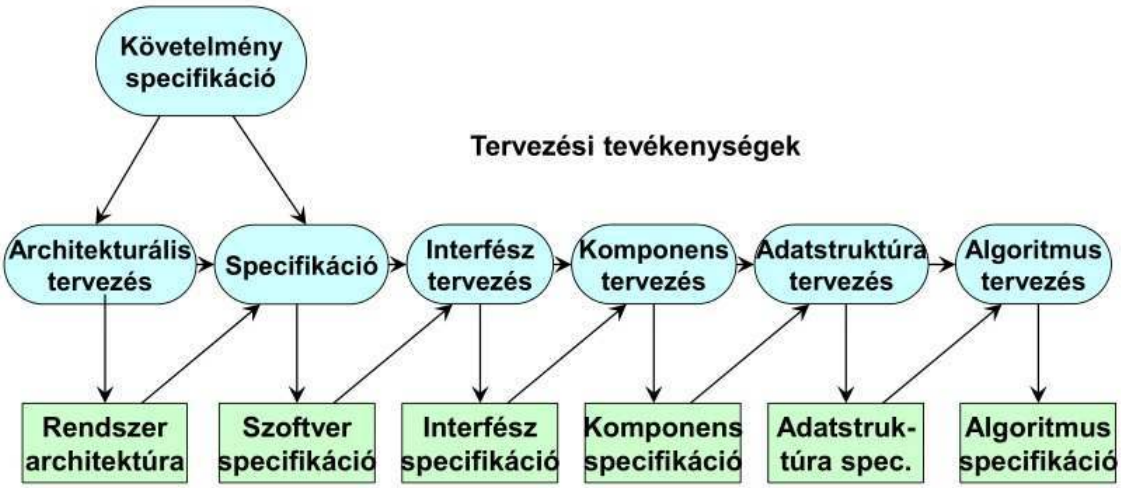
- Azoknál a rendszereknél, amelyeket formális módszerekkel fejlesztettek, a hibaarány jóval alacsonyabb volt.
- Ezért elsősorban kritikus rendszerek fejlesztésénél alkalmazzák, ahol a rendszer igen magas verifikálási és validálási költségeihez és az esetleges hibákból eredő katasztrófa költségeihez képest még kifizetődő a használata.

Milyen típusú rendszereknél alkalmazzák?
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. A formális specifikációt nyílt interfészek, kommunikációs protokollok definiálására is alkalmazzák.

34. Melyek azok a rendszerek, amelyeknél a formális specifikációt leginkább alkalmazzák? Miért?

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

35. Hol foglal helyet az architekturális tervezés a szoftverfolyamatban? Mire szolgál a rendszer architekturális terve?



Mire szolgál a rendszer architekturális terve?
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. Ez a rendszertervezés folyamatának kezdeti lépcsőfoka.
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 tevékenységekkel párhuzamosan végezhető. A kezdeti architekturális elképzelések már a specifikációban tükröződhetnek. Magába foglalja a fő rendszerkomponensek és azok vezérlésének, valamint kommunikációjának meghatározását.

36. Kliens/szerver modell

Olyan osztott rendszermodell, amely bemutatja hogyan oszlanak meg az adatok és a feldolgozások a komponensek között.

Elemei:

- **Szerverek:** Adatkezelő szerverek, nyomtatószerverek, alkalmazás szerverek, kommunikációs szerverek, stb.
- **Kliensek:** Többnyire önálló alrendszerek, amelyek hozzáférnek a szerverek szolgáltatásaihoz. Egyszerre sok példányban futnak.
  Típusai:
  - Vékony kliens: Böngésző, szkriptekkel
  - Vastag kliens: komplett kis alrendszer, helyi adatokkal, funkciókkal
- **Hálózat:** A klienseknek biztosít hozzáférést a szerverek szolgáltatásaihoz.

37. Milyen előnyei és hátrányai vannak a kliens/szerver modellnek?

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 szerint kialakított adatmodellt használja (ez előny a teljesítmény szempontjából).
- Redundáns adatkezelés folyhat minden szerverben.
- Nincs központi név- és szolgáltatás nyilvántartás, nehéz megtalálni, hogy milyen szerverek és szolgáltatások léteznek.

38. Mi a különbség a vékony- és a vastag kliens között? Melyik milyen célra alkalmas?

Vékony kliens: A vékony kliens egy minimális eszközökkel rendelkező kliens. Ez a kliens típus a szükséges erőforrásokat is a távoli (host) gépen veszi igénybe. Egy vékony kliens feladata többnyire kimerül az alkalmazásszerver által küldött adatok grafikus megjelenítésében; a tényleges, nagy mennyiségű adat mozgatását, kezelését igénylő feladatot az alkalmazás szerver végzi el. (Pl.: Web böngészőből futó alkalmazások)
Vastag kliens: A vastag kliens képes arra, hogy önmaga hajtson végre nagyobb adatmennyiségekkel feldolgozásokat, amikor a szerver inkább elsődleges tárolóként viselkedik. Ennek ellenére, a kifejezés inkább a számítógép szoftverére vonatkozik, és egyre inkább alkalmazzák hálózati számítógépek esetén, ahol a számítógép jelentős hálózati alkalmazásokat (is) futtat.

39. Milyen modelleket alkalmaznak az objektumorientált tervezésben? Melyik mire alkalmas?

Objektum-modellek:

- A rendszer felbontása együttműködő objektumokra. Az objektumok egyéni állapottal és az állapotokon értelmezett műveletekkel rendelkeznek.
- A rendszert jól definiált interfészekkel rendelkező, lazán csatolt objektumokra bontja, amelyek egymás szolgáltatásait veszik igénybe.
- Az objektum orientált felbontás az objektum osztályok, attribútumaik és műveleteik azonosítását (felismerését és helyes modellezését) jelenti.
- Az implementáció során a konkrét objektumok ezekből az osztályokból jönnek létre. Az objektumok műveleteinek koordinálását valamilyen vezérlési modellel ábrázolják.

Adatfolyam-modellek:

- A rendszer felbontása funkcionális modulokra, amelyek az inputokat outputokká transzformálják (csővezeték modellnek is nevezik). A modulok funkcionális transzformációk.
- Az adatfolyam modellben az inputot funkcionális transzformációk dolgozzák fel és ennek eredményeként állítják elő az outputot.
- Tulajdonképpen megegyezik a UNIX shell „pipe and filter" modelljével.
- Régóta alkalmazzák az adatfeldolgozási rendszerek modellezésére. (főleg kötegelt, szekvenciális adatfeldolgozás esetén)
- Interaktív rendszerek modellezésére csak nagyon precíz specifikálással alkalmas.

40. Mi a különbség a központosított vezérlés és az esemény alapú vezérlési modell között?

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

Esemény alapú vezérlés:
Minden alrendszer reagálhat az őt érintő külső, vagy más alrendszer által generált eseményekre. 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 megszakításkezelő észleli az eseményt és elindítja az esemény feldolgozásáért felelős alrendszert.

41. Milyen vezérlési modellek alkalmazhatók párhuzamos rendszerekben?

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

42. Milyen UML modellekkel ábrázolható a rendszer és környezetének kapcsolata?

Környezeti modell:

- 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 rendszerek közti kapcsolatok ábrázolása ugyancsak a környezeti modellek feladata.
- A környezeti modell ábrázolási módja általában egyszerű blokkdiagram.

43. Mire szolgálnak az objektumorientált tervezésben alkalmazható diagramok? Soroljon fel és jellemezzen néhányat!

- **Alrendszer modellek:** Az objektumok logikai csoportosítását mutatják az összefüggő alrendszerekben.
- **Szekvencia modellek:** Az objektumok interakcióinak sorrendjét ábrázolják.
- **Állapotmodellek:** Bemutatják, hogy egy objektum hogyan változtatja az állapotát, válaszol bizonyos eseményekre.
- **Egyéb modellek:** Használati eset modellek, öröklődési modellek, osztálydiagramok, stb.

44. Ismertesse a valós idejű rendszerek főbb jellegzetességeit.

A valós idejű rendszerek olyan (gyakran beépülő) szoftverrendszerek, amelyek figyelik környezetüket és adott (rövid) időn belül képesek reagálni a környezeti hatásokra (ingerekre). Általában inger-válasz típusú rendszerek.
Vannak:

- **Periodikus ingerek:** Időzítés hatására végez valamit a rendszer.
- **Aperiodikus ingerek:** Rendszertelenül bekövetkező külső inger hatására kell valamit csinálni.

A valós idejű rendszerek működésében az idő kritikus tényező.
A valós idejű rendszerek gyakran az átlagosnál nagyobb felelősségű feladatot látnak el.
A valós idejű rendszerekhez mindig tartoznak hardver eszközök is:

- Érzékelők, amelyek adatokat gyűjtenek.
- Szabályozók, működtetők, amelyek a rendszer környezetét befolyásolják.

45. Melyek a fő különbségek az átlagos adatfeldolgozó rendszerek és a valós idejű rendszerek között?

Míg a korábban kialakult adatfeldolgozó rendszerek tervezése általában az adatszerkezetek vagy az adattransz-formációk (funkciók) megragadásával kezdődik, addig az ilyen célrendszerek(valós idejű rendszerek) tervezői leggyakrabban a rendszert érő hatások (külső események) és az erre adandó válaszok – azaz a rendszer viselkedése – elemzéséből indulnak ki. A viselkedés megadásakor általában nem elegendő azt előírni, hogy a rendszernek milyen külső események hatására milyen válaszokat kell előállítania, hanem a számítógépes rendszer válaszidejére is megkötéseket kell tenni.

46. Van-e szerepe a valós idejű rendszerek tervezésében a hardvertervezésnek? Miért?

A rendszer hardver és a szoftver elemeit együtt kell megtervezni, célszerűen elosztva a funkciókat a hardver és a szoftver között. A döntést azonban, hogy mit kell hardverben és mit szoftverben megvalósítani, célszerű halogatni, az optimális megoldás megtalálása érdekében. Egy funkció sok esetben hardverrel jobb teljesítménnyel valósítható meg, de hosszabb fejlesztést igényel és a változások nehezebben követhetők.

47. Milyen programnyelveket alkalmaznak a valósidejű rendszerek programozására?

- **C:** Lehetséges effektív programokat írni, de nem feltétlenül támogatja a párhuzamos folyamatokat, vagy a megosztott erőforrások kezelését. Ezeket azonban az operációs rendszer megoldhatja.

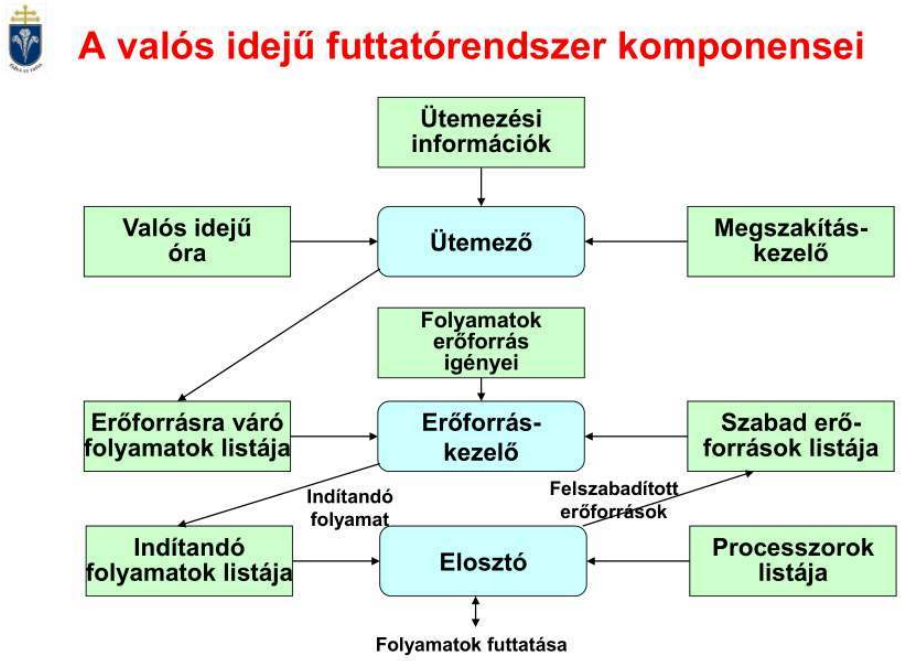
- **Ada:** Valós idejű rendszerek programozására készült, ezért támogatja a konkurenciát és az újabb verziói már az ütemezést és az időzítést is kezelik.
- **JAVA:** A Java támogatja a konkurenciát (szálak és szinkronizált módszerek), ezért alkalmas a kevéssé kritikusan valós idejű rendszerek fejlesztésére.

48. Miért kevéssé alkalmas a Java programozási nyelv szigorúan valós idejű rendszerek programozására?

Nem használható szigorúan real-time rendszerekhez, mert:

- Nem lehet megadni egy szál végrehajtási idejét.
- A „szemétgyűjtés” nem vezérelhető.
- A megosztott erőforrásokat tartalmazó sorok méretét nem lehet lekérdezni.
- A különböző virtuális gép implementációk különböző időzítéssel futtatják ugyanazt a szoftvert.
- A futási idő tár- és processzorhasználatának elemzése nem lehetséges.

49. Melyek a valós idejű futtatórendszerek főbb komponensei?



50. Melyek a szoftver újrafelhasználásán alapuló fejlesztés előnyei és hátrányai?

Előnyök:

- **Javuló megbízhatóság:** A komponenseket már több működő rendszerben kipróbálták.
- **Alacsonyabb projektkockázat:** A komponensek ára és adaptálási költsége pontosabban tervezhető.
- **A szaktudás jobb kihasználása:** A speciális szaktudás a komponensben testesül meg, nem szükséges minden projekthez külön alkalmazni.
- **Szabványosság:** A szabványoknak való megfelelést a komponensek garantálják. (interfészek, kommunikációs és GUI szabványok)
- **Gyorsabb fejlesztés:** Egy rendszer kifejlesztése gyorsabb, ha kevesebb eredeti fejlesztést igényel.

Hátrányok:

- **Növekvő karbantartási költségek:** A komponens forráskódja és tervezési dokumentációja hiányában növekszik a karbantartás költsége.
- **Az eszköztámogatás hiánya:** A CASE eszközök gyakran nem támogatják az újrafelhasználást.
- **A "nem mi találtuk ki" jelenség:** Egy teljes rendszer kidolgozása nagyobb szakmai kihívás.
- **A komponenskönyvtárak karbantartása:** Sokba kerül a komponenskönyvtárak feltöltése és folyamatos karbantartása.
- **Az újrafelhasználható komponensek megtalálása és adaptálása:** Még nem fejlődtek ki a komponensek megtalálását és adaptálását segítő általános technikák.

51. Soroljon fel három érvet, amely támogatja a komponens alapú újrafelhasználást és hármat, amely ellene szól!

Előnyök:

1. A komponens alapú fejlesztés beilleszthető a szabályos szoftverfolyamatba.
2. A komponensek az objektumosztályoknál sokkal absztraktabbak és különálló szolgáltatásoknak tekinthetők.
3. A legtöbb programozási nyelvben hivatkozhatunk könyvtárban tárolt komponensekre.

Hátrányok:

1. A komponensek inkompatibilitása miatt a költség- és időmegtakarítás a vártnál kevesebb lehet (integrációs munkák).
2. Nehézséget okozhat a komponensek megtalálása (nincsenek szabványok a komponensek tulajdonságainak leírására, hiányoznak az egységes komponens könyvtárak).
3. A követelmények változását követő evolúció lehetetlen, ha a komponensek nem cserélhetők.

52. Mi a programgenerátor alapú újrafelhasználás lényege? Milyen területeken alkalmazzák?

A generátor alapú újrafelhasználás akkor lehetséges, ha egy programgenerátor tartalmazza egy szakterület alapvető ismeretanyagát. (pl. adatfeldolgozás). Az ilyen programgenerátorok tartalmazzák a szabványos algoritmusokat és függvényeket, és ezek paraméterezését követően a generátor automatikusan előállítja a programot. A szakterületre kidolgozott nyelven, vagy újabban grafikus eszközökkel lehet elkészíteni a rendszer modelljét.

53. Miért alkalmazzák a generátor alapú újrafelhasználást elsősorban az adatfeldolgozó, eBusiness rendszerek fejlesztésében?

A generátor alapú újrafelhasználás igen költséghatékony, de viszonylag kevés szakterülethez léteznek ilyen rendszerek.(pl.: adatfeldolgozó, e-business).
Ezzel a módszerrel könnyebben állíthatók elő az alkalmazások, mint a komponens alapú módszerrel.

54. Mi a komponens, milyen interfészei vannak?

A komponensek szolgáltatásokat nyújtanak a rendszer számára, a végrehajtás helyétől és a megvalósítás nyelvétől függetlenül.

- 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 sok esetben nem hozzáférhető, belső állapotai nem láthatóak.

Komponens interfészei:

- **Szolgáltatott interfészek:** A komponens által szolgáltatott interfészek.
- **Szükséges interfészek:** Azok az interfészek, amelyeket a komponenst használó rendszernek, vagy környezetének kell biztosítania.

55. Milyen nyelvek és környezetek alkalmasak a komponensek integrálására?

A legtöbb programnyelvben hivatkozhatunk könyvtárban tárolt komponensekre. Leggyakrabban szkriptnyelveket használnak komponensek integrálására.
Köztes integrációs keretrendszerek: komponensek közti kommunikációt és információcserét támogató szabványok és osztályok. (Ilyen például a CORBA, JavaBean, COM, DCOM, stb.)

56. 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.

57. Mire szolgának az alkalmazási keretrendszerek, hogyan csoportosíthatók?

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 keretrendszerek csoportosítása:

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

58. Mi a „polcról levehető” termék? Leginkább milyen rendszerekben alkalmazzák?

A „polcról levehető”, COTS – Commercial Off-The-Shelf rendszerek általában komplett alkalmazási rendszerek, amelyek API-val rendelkeznek. Legtöbbször rendszerszoftver termékek, az egyszerű komponenseknél nagyobb funkcionalitással.
Nagy rendszerek építésekor gyakran használt stratégia a COTS termékek integrálása. Különösen a gyors fejlesztést kívánó eCommerce, eBusiness rendszerek körében. A fejlesztési idő nagyságrendekkel csökkenthető.

59. Milyen nehézségei vannak a COTS termékek alkalmazásának?

- **A funkcionalitás és a teljesítmény nem tartható kézben:** A COTS rendszerek sokszor kevésbé effektívek mint azt a reklámokban ígérik.
- **A COTS rendszerek együttműködése bizonytalan:** A különböző COTS rendszerek eltérő feltételezésekkel készültek (pl. sorkezelés), ezért az integráció nehéz lehet.
- **Az evolúció ellenőrizhetetlen:** A szállító és nem a felhasználó határozza meg.
- **A COTS termékek támogatása:** A szállító által biztosított támogatás gyakran nem terjed ki a rendszer teljes élettartamára.

60. Miért drágább egy újrafelhasználható komponens kifejlesztése az egyedi komponens kifejlesztésénél?

- Az újrafelhasználható komponensek meglévő komponensek általánosításával hozhatók létre. Ehhez nagy tapasztalat kell(tehát hozzáértő személy).
- 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.

61. Mi az alkalmazáscsalád? Miért alakultak ki az alkalmazáscsaládok?

Alkalmazás család: Az alkalmazáscsalád az alkalmazási rendszerek olyan termékcsaládja, vagy termékvonulata, amelyek egy közös, szakterület-specifikus architektúrára épülnek („közös mag").
Kialakulásának célja: A kialakulás célja az volt, hogy az alkalmazáscsalád magját (szoftver magját) újra feltudják használni, amikor egy új alkalmazást szeretnénk fejleszteni.

62. Hogyan osztályozható az alkalmazáscsaládok specializációja?

- **Platform specializáció:** Az alkalmazás egyes verziói különböző platformokra készülnek (pl. Windows, Solaris, Linux, ..), de a funkcionalitás azonos.
- **Konfigurációs specializáció** Az alkalmazás egyes verziói különböző perifériákkal képesek együttműködni.(A perifériákat kezelő komponensek különböznek.)
- **Funkcionális specializáció** Az alkalmazás egyes verziói eltérő funkcionális követelmények kielégítésére készülnek.(A funkcionális komponensek különböznek.)

63. Mi a különbség a tesztelés és a belövés között? Melyiknek mi a célja?

Különbségek:

- 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.
- A belövés során felfedezett hibák javítása után újra kell tesztelni a programot.

Tesztelés célja:

- Felfedni a rendszerben (esetleg már a tervek szintjén) rejlő hibákat, nem arról megbizonyosodni, hogy hibamentes a rendszer.
- Meggyőződni arról, hogy a rendszer egy-egy konkrét működési szituációban használhatóan működik.

Belövés célja:

- A felfedett hibák helyének lokalizálása és kijavítása.

64. Mi a célja a szoftver verifikációjának és mi a validáció feladata?

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.

65. Miért célszerű a szoftvertesztelés mellett átvizsgálást (inspekció) is tartani?

Szoftver-átvizsgálás (inspekció): A rendszer reprezentációjának elemzése (Követelményspecifikáció, tervek, grafikus ábrázolások, forráskód). A forráskód elemzése automatizálható.(Statikus verifikáció)
Miért célszerű inspekciót is tartani?

- 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.(Legalábbis, ha egy hibát detektál, a tesztelést abba kell hagyni, és a hiba kijavítását követően újból elölről kell kezdeni.)
- Sok költséges tesztelést előzhet meg.

66. Mi a programtesztelés feladata? Milyen alaptípusai vannak?

Szoftvertesztelés: A szoftver implementációjának tesztadatokkal való futtatása és a viselkedés megfigyelése(Dinamikus verifikáció).
Típusai:

- **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, stressz-tesztek
    - **Objektumorientált tesztelés**
- **Statisztikai tesztelés:** A rendszer teljesítményének és megbízhatóságának tesztelése, valós helyzetekben (valós felhasználói inputtal és gyakorisággal).

67. Mire szolgál a szoftver átvizsgálása? Mi a különbség az átvizsgálás és a tesztelés között?

A szoftver átvizsgálásának célja: A szoftver átvizsgálás célja a hiányosságok felderítése, a költséges tesztelés helyett a hibák kb. 60 %-a felfedhető az átvizsgálás során.
Különbségek:

- Egy átvizsgálás során több hiányosság felfedezhető, amíg egy teszt többnyire egy hibát fed fel.(Legalábbis, ha egy hibát detektál, a tesztelést abba kell hagyni, és a hiba kijavítását követően újból elölről kell kezdeni.)
- Az átvizsgálás már fejlesztési folyamat kezdetétől alkalmazható, a dokumentumok (követelmények, tervek) átvizsgálásával, míg a tesztek nem.
- Az inspekció alkalmas eszköz arra, hogy ellenőrizze, megfelel-e a program a specifikációnak.

68. Milyen hiányosságokat lehet elsősorban felfedezni a programok átvizsgálása során?

- Dokumentumokban és forráskódban rejlő hiányosságok.
- Követelményekben és tervekben lévő hiányosságok felderítése.

69. Mi a Cleanroom szoftverfejlesztési folyamat lényege?

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

70. Ismertesse a grafikus felhasználói kezelőfelületek tervezésének alapelveit!

Alapelvek:

- **A felhasználói jártasság figyelembevétele:** A felületnek olyan kifejezéseket és fogalmakat kell használnia, amelyeket az átlagos felhasználó ismer.
- **A felület konzisztenciája:** Azonos menüknek és parancsoknak azonos formátummal kell rendelkezniük, hasonló műveleteket hasonló módon és helyen kell jelezni és megvalósítani.
- **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”
- **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 felhasználók sokfélesége:** Az alkalmi felhasználók több támogatást, a gyakorlott felhasználók egyszerűbb, gyorsabb működést várnak.

71. Miért fontos a grafikus felhasználói kezelőfelület gondos tervezése?

- 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 gyakran a kezelőfelület megtervezése is a szoftver tervező feladata.

72. Miért célszerű egyes rendszerekben különböző felhasználói felületeket kidolgozni a gyakorlott és az alkalmi felhasználók számára?

Az alkalmi felhasználók több támogatást, a gyakorlott felhasználók egyszerűbb, gyorsabb működést várnak. Az alkalmi felhasználók inkább az ablakos, kattintgatós felhasználós felületeket szeretik, vagy a természetes nyelven működő parancssoros alkalmazásokat, mivel ezek megtanulása könnyebb. Míg a gyakorlottabb felhasználók inkább a terminálos (parancssoros), vagy menüs felhasználói felületeket kedvelik a gyors művelet végrehajtás miatt.

73. Milyen alapelemeket használunk a grafikus felhasználói kezelőfelületeken?

- **Ablakok:** Az ablaktechnikával több ablakban egyszerre többféle információ jeleníthető meg.
- **Ikonok:** Az ikonokkal az információ fajtái jelölhetők: állományok, folyamatok, stb.
- **Menük:** A menütechnikával a parancsok egy strukturált menüből választhatók ki. A felhasználónak nem kell egy parancsnyelvet megtanulnia és parancsokat begépelnie.
- **Pozicionálás:** Egér, vagy más eszköz alkalmazható egy menüpont kiválasztására, vagy egy ablakban a lényeges elemek meg- vagy kijelölésére.
- **Grafika(színek, képek):** Grafikus elemek és színek alkalmazása a szöveg mellett (vagy helyett) áttekinthetőbbé teszi a képernyőt.

74. Milyen eszközöket alkalmazhatunk a grafikus felhasználói kezelőfelületen a felhasználó támogatására?

A kezelőfelület tervezésekor figyelembe kell venni a felhasználók igényeit, gyakorlatát és képességeit. 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.

75. Milyen előnyei és hátrányai vannak a parancsnyelv alkalmazásának a felhasználói interakciókban?

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

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

Parancsnyelv hátrányai:

- Nehezen tanulható, az átlagos felhasználó számára bonyolult.
- Gépelési gyakorlatot kíván.
- A hibakezelést (hibajelzés, visszavonás) nehéz megoldani.

A parancsnyelveket a gyakorlott felhasználó számára lehet alkalmazni. A menürendszer alternatívájaként célszerű biztosítani.

76. Sorolja fel és jellemezze a felhasználói interakciók fajtáit!

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

- **Előnyei:**
  - Könnyen tanulható és gyors,
  - A felhasználó azonnal visszajelzést kap, így a tévedés gyorsan visszavonható.
- **Hátrányai:**
  - Bonyolult lehet a felhasználó tevékenységéről (szándékáról) a megfelelő információt begyűjteni a program számára,
  - Csak akkor használható, ha a feladatok és objektumok egyértelműen megkülönböztethető ikonokkal reprezentálhatók.

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:**
  - A felhasználónak nem kell parancsokat megjegyeznie,
  - Kevés gépelést igényel és a hibák könnyen kivédhetők,
  - Állapotfüggő súgó alkalmazható.

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

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

- **Előnyei:**
  - A felhasználói hibák felfedhetők és jelezhetők, illetve kivédhetők,
  - Legördülő választási lehetőséggel sok felhasználói tévedés kizárható,
  - Könnyen megtanulható.
- **Hátrányai:**
  - Nagy képernyőfelületet foglal.

Parancsnyelv: A felhasználó parancsokat gépelve utasítja a rendszert (pl. Unix). A parancsnyelveket a gyakorlott felhasználó számára lehet alkalmazni. A menürendszer alternatívájaként célszerű biztosítani.

- **Előnyei:**
  - Egyszerű, olcsó terminálon is alkalmazható,
  - Egyszerűen feldolgozható (pl. fordító technikával)
  - Bonyolult, egymásba ágyazott parancsok is kezelhetők,
  - Rugalmas.
- **Hátrányai:**
  - Nehezen tanulható, az átlagos felhasználó számára bonyolult.
  - Gépelési gyakorlatot kíván.
  - A hibakezelést (hibajelzés, visszavonás) nehéz megoldani.

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.
- Beszédfelismeréssel kombinálva – szűkített kifejezésekkel – ma is használják.

77. Milyen lehetőségek vannak az információ grafikus megjelenítésére?

- 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 megjelenítését.

Az információ lehet **statikus információ:**

- Értéket kap a munkafázis (session) kezdetén és ez a session ideje alatt nem változik meg,
- Lehet numerikus, szöveges, vagy grafikus.

Vagy **dinamikus információ:**

- Megváltozik a munkafázis alatt és a megváltozott értéket a felhasználó számára meg kell jeleníteni,
- Lehet numerikus, szöveges, vagy grafikus.

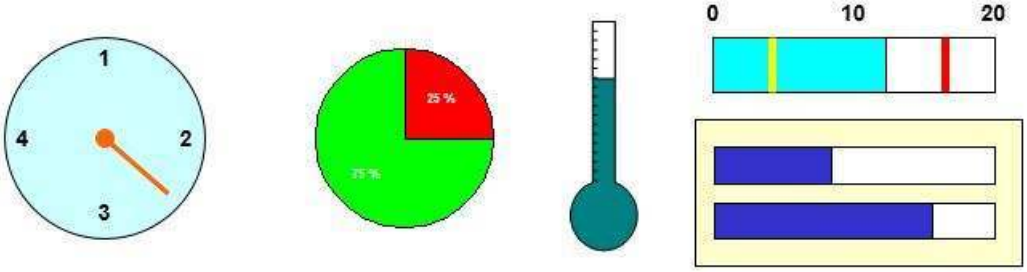
78. Írjon példákat, amikor az információt célszerű grafikusan, analóg módon megjeleníteni.

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



79. Milyen szabályokat kell betartani a színek alkalmazásakor a grafikus felhasználói kezelőfelületen?

- Ne használjunk túl sok színt. Egy felületen 4-5, egy rendszerben 7-8 szín a maximum.
- Először tervezzünk monokróm felületeket, utána adjuk hozzá a színeket.
- Az állapotváltozásokat jelezzük színváltással.
- A végrehajtandó feladatokat jelöljük színkóddal, a különböző feladatokat különböztessük meg színekkel is.
- A színkódolást alkalmazzuk következetesen a teljes rendszerben.
- Egyes színkombinációk zavaróak, vagy fárasztják a szemet.

80. Milyen szempontokat kell figyelembe venni a rendszer üzeneteinek szövegezésekor?

Szempontok:

- 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 hibaüzenetek tervezésének meghatározó tényezője a felhasználó háttere, gyakorlata.

Az üzenetek szövegezése:

- **Szövegkörnyezet:** A tájékoztató rendszernek mindig a felhasználó tevékenységéhez és a rendszer aktuális állapotához igazodó üzenetet kell adnia.
- **Tapasztalat:** A tapasztalt felhasználót már idegesíti az a kifejtő magyarázat, amit a kezdő felhasználó még hasznosnak tart és igényel. A tájékoztató rendszernek mindkét üzenettípust fel kell kínálnia.
- **Képzettség:** Az üzeneteket a felhasználó képzettségéhez és gyakorlatához kell igazítani. A különböző felhasználók számára szánt üzeneteket különböző módon, a számára érthető terminológiával kell megfogalmazni.
- **Stílus:** Az üzeneteket pozitív módon építő jelleggel kell megfogalmazni. Egy üzenet soha nem lehet sértő és nem gúnyolódhat.
- **Kultúra:** Hasznos, ha az üzenetek tervezője tisztában van azzal a kultúrával, ahol a rendszert használni fogják. Az egyik országban megfelelő üzenetek a kulturális különbségek miatt egy másik országban elfogadhatatlanok.

81. Miért célszerű a súgórendszerbe több belépési pontot biztosítani?

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.

A súgórendszer használata:

- 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.

82. Mi a teszteset és a tesztadat? Hogyan lehet a tesztadatok számát csökkenteni?

Teszteset: A tesztesetek a teszthez szükséges inputok és a várt outputok specifikációi.
Tesztadat: A tesztadatok a rendszer tesztelésére kidolgozott input adatok.
Tesztadatok számának csökkentése: Ekvivalenciaosztályok segítségével ?

83. Mi a „fekete doboz” és a „fehér doboz” tesztelési stratégia lényege? Melyiket milyen esetben lehet alkalmazni?

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.(Egyes Agilis módszereknél előbb, mint a program tervezése!)
- Az előreláthatóan hibát okozó tesztesetek tervezéséhez szakterületi ismeretekre van szükség.

Fehér doboz tesztelés (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).

Alkalmazási területek:

- Az objektumokhoz kapcsolódó műveletek tesztelése: Függvények, vagy eljárások, fekete- vagy fehér doboz eljárással tesztelhetők.
- Fekete doboz tesztek: integrációs teszt, objektumosztályok tesztelése.

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

Ekvivalencia-osztályozás:

- A rendszer input és output adatait valamilyen közös jellegzetesség szerint csoportosítják, amelyekre a rendszer hasonló módon reagál.
- A fejlesztők legtöbbször az inputok tipikus értékeit veszik figyelembe.
- A teszteseteket a határértékek közelében és az osztályok közepéből célszerű kiválasztani.

Példa: Ha az input 5 jegyű valós szám 10.000 és 99.000 között, akkor az ekvivalencia-osztályok:

- Azok a számok melyek kisebb 10.000-nél
- Azok a számok melyek 10.001-99.000 között vannak
- Azok a számok melyek 99.900-99.999 között vannak
- Azok a számok melyek 99.999-nél nagyobbak

85. Mi a célja az útvonaltesztelésnek? Mi a ciklomatikus komplexitás, hogyan számítható?

Útvonal tesztelés célja:

- 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 folyamat-gráfja, 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.

Ciklomatikus komplexitás:

- A független utak száma a programban.
- 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.
- A dinamikus programelemzők a fordításkor kiegészítő kódot adnak a programhoz, amelyek mérik, hogy az egyes vezérlő utasítások hányszor kerültek végrehajtásra.

Ciklomatikus komplexitás számolása:

$$CC = Élekszáma - Csomópontokszáma + 2$$

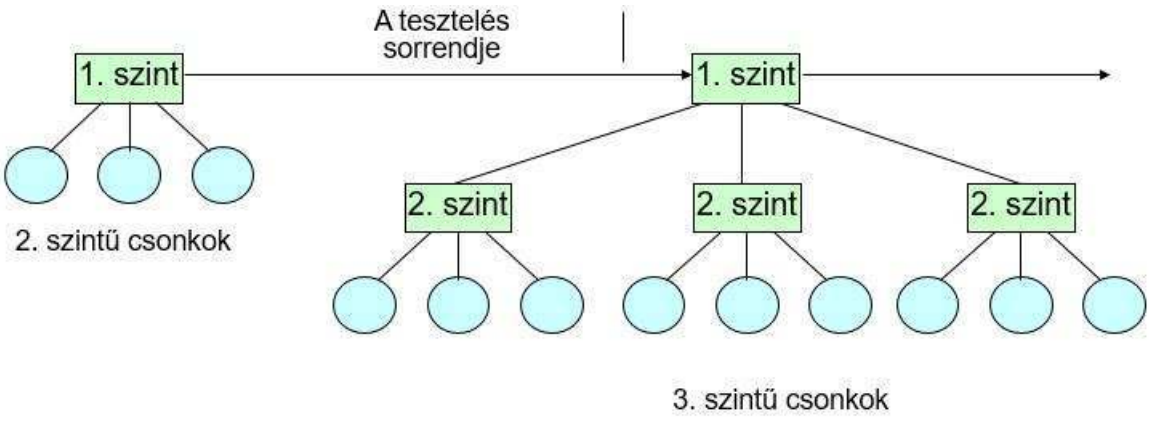
86. Ismertesse az integrációs tesztelési stratégiákat! Mi az összefüggés e stratégiák és a szoftverfolyamat modellje között?

Integrációs tesztelés:

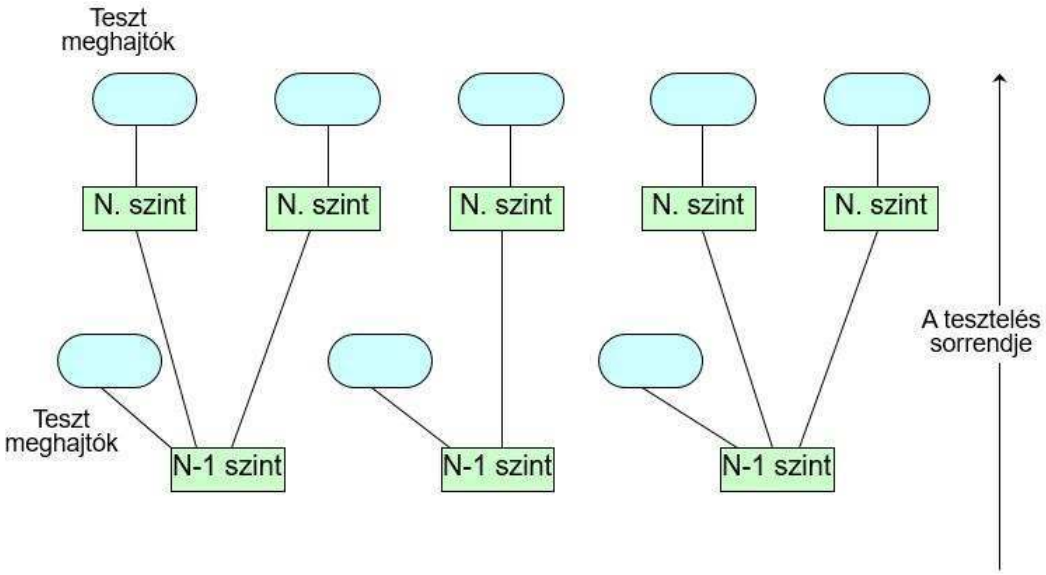
- Teljes rendszerek vagy alrendszerek tesztelése, amelyek előzőleg már tesztelt komponensekből állnak.
- A komponensek együttműködéséből származó hibák feltárására szolgál.
- Az integrációs teszt fekete doboz tesztelés, a tesztek a specifikációból származnak.
- Komplex rendszerben az észlelt hibás eredményből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

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 alkalmazható)



A gyakorlatban a kettő kombinációját használják.
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 sokszor nem szolgáltatnak outputokat.

87. Mi az interfésztesztelés, milyen hibákat lehet felfedni ilyen módon?

Interfésztesztelés:

- Interfésztesztelésre akkor van szükség, amikor egy nagyobb rendszer összeépítésekor modulokat vagy alrendszereket integrálunk.
- Az interfésztesztelés az objektumorientált fejlesztésnél fontos (különösen objektumok és osztályok újrafelhasználásakor), mert az objektumokat az interfészeikkel definiáljuk.

Felderíthető hibák:

- Célja az interfészek specifikációs- (félreértések), vagy implementációs hibáinak felfedése.

Nem felderíthető hibák:

- Egyedi objektum tesztelésével az interfészhibákat nem lehet felfedni. A hibák az objektumok közti interakciókban jelentkeznek, nem egy egyedi objektum sajátosságaiként.

88. Melyek a tipikus interfészhibák? Milyen elveket kell alkalmazni az interfésztesztelés tervezésekor?

Tipikus interfészhibák:

- **Interfész hibás alkalmazása:**
  Egy hívó komponens hibája lehet: rossz típusú vagy sorrendű paraméterek, hibás számú paraméter, stb.
- **Interfész félreértése:**
  A hívó komponens hibásan értelmezi az interfészt, vagy a hívott komponens válaszait.
- **Időzítési hibák:**
  A hívó és a hívott komponens különböző sebességgel működik (osztott memória, vagy üzenettovábbító interfész esetén), és a hívott nem aktuális információt kap.

Interfésztesztelés irányelvei:

- A teszteket úgy kell tervezni, hogy a paraméterek értékei a határértékek közelében legyenek.
- A pointer jellegű paramétereket null értékkel is tesztelni kell.
- Olyan tesztesetet is tervezni kell, amely a hívott komponens hibáját okozza. (A specifikációs hibák többsége a hibák értelmezéséből fakad.)
- Üzenettovábbító, vagy interaktív rendszereknél terhelési (stressz) tesztet kell végrehajtani.
- Osztott memóriájú interfészeket a komponensek aktiválódása sorrendjének megváltoztatásával is tesztelni kell (szinkronizációs hibák).

89. Miért és miben különbözik az objektumorientált tesztelés a funkcióorientált rendszerek tesztelésétől?

Objektumorientált tesztelés:

- A komponens- és integrációs tesztelés az objektumorientált rendszereknél is alkalmazható.
- **Fontos különbségek:**
  - A tesztelendő objektumok komponensként gyakran nagyobbak, mint az egyszerű függvények.(A fehér doboz tesztelés nehezebben alkalmazható.)
  - Az objektumok lazán kötődnek, és a rendszernek/alrendszernek nincs egyértelmű teteje.
  - Az újrafelhasznált komponensek kódjához nem mindig lehet hozzájutni, elemezni.

Megkülönböztetések:

- **A funkcióorientált rendszereknél:**
  - A rendszer alapvető program-egységei (függvények – modulok) jól elkülöníthetők,
  - Ezek külön tesztelhetők.
- **Az objektumorientált rendszerek esetén:**
  - Az ilyen megkülönböztetés nem lehetséges, az objektumok lehetnek egyszerű (pl. lista), vagy komplex entitások (pl. egy alrendszer objektumai),
  - Olykor nincs egyértelmű hierarchia az objektumok között, ezért az integrációs tesztek (fentről lefelé, vagy lentről felfelé) nem alkalmazhatók.

90. Milyen szinteket különböztethetünk meg az objektumorientált tesztelésben?

- **Az objektumokhoz kapcsolódó műveletek tesztelése:**
  Függvények, vagy eljárások, fekete- vagy fehér doboz eljárással tesztelhetők.
- **Objektumosztályok tesztelése:**
  A fekete doboz eljárás alkalmazható, de az ekvivalencia-osztályokat a műveletsorozatokra is ki kell terjeszteni.
- **Együttműködő objektumcsoportok tesztelése:**
  Forgatókönyv alapján kijelölhető az objektumok csoportja.
- **Objektumorientált rendszer tesztelése:**
  A rendszerkövetelmények verifikációja és validációja más rendszerekhez hasonlóan történhet.

91. Mi az objektumorientált csoporttesztelés? A rendszerterv milyen elemeit lehet felhasználni a csoporttesztéléshez?

Objektumorientált csoporttesztelés:

- **Használati eset vagy forgatókönyv alapján:** A tesztek a felhasználói interakciókon alapulnak. Előnye, hogy a felhasználók által leggyakrabban használt részeket teszteli.
- **Száltesztelés:** A rendszernek egy eseményre adott válaszát vizsgálja, amint az a rendszeren keresztülhalad.
- **Objektum együttműködési teszt:** Az objektumok együttműködésének egy sorozatát vizsgálja, amely akkor ér véget, ha egy objektumművelet nem hív meg más objektumszolgáltatást.

Elemek:

- Forgatókönyv alapján kijelölhető az objektumok csoportja.

92. Mi a szoftver költségbecslés célja? Milyen kérdésekre keresi a választ?

Kérdések:

- Mekkora munkát igényel egy feladat elvégzése?
- Mennyi időbe kerül a feladat végrehajtása?
- Mennyi a tevékenység összes költsége?

Költségbecslés célja: Választ adni a következő költségelemekre a szoftverfejlesztési projekt során:

- A hardver és szoftver költségei a karbantartással együtt,
- Utazási és képzési költségek,
- Munkaköltségek (bér, közteher, helység, kisegítő munkák, kommunikáció, rekreáció, ...)

93. Milyen módszereket ismer a szoftver költségének előzetes becslésére?

Funkciópontok:

- A program jellemzőinek kombinációján alapuló, nyelv-független módszer.
- **Méri az alábbi jellemzőket:**
  - Külső bemenetek és kimenetek
  - Felhasználói interakciók
  - Külső interfészek
  - A rendszer által használt fájlok
- **Mindegyikhez súlyozási tényezőt rendel:**
  - Egyszerű külső bemenet: 3
  - Bonyolult belső állományok: 15
- A súlyozási tényezőt egy szervezeten belül, hasonló jellegű szoftverek készítése során gyűjtött statisztikák alapján finomítja.

A funkciópontok számítása:

- A funkciópontok (FP) alapján a kódsorok számára (LOC – Lines Of Code) lehet következtetni:
- LOC = AVC * FP ahol:
  AVC nyelvfüggő szorzófaktor (200-300 az assembly és 2-40 a 4GL nyelvekre)
- A funkciópont számítás nagyon sok szubjektív elemet tartalmaz.
- Automatikus számítása nem lehetséges, mert a specifikáció alapján kell a funkciópontokat megbecsülni.

Objektumpontok:

- 4GL vagy más magas szintű nyelvek esetén a funkciópontok alternatívája. Magas szintű specifikáció alapján könnyebben becsülhető.
- Az objektumpont (NTC) nem azonos az objektumok számával, hanem az alábbiakból számítható:
  - A külön megjelenítendő képernyők száma, az egyszerűtől (1), a nagyon bonyolultig (3),
  - A készítendő jelentések száma (2 – 5 – 8)
  - A 4GL kiegészítése miatt szükséges 3GL modulok száma (10)

A termelékenység becslése:

- Valósidejű, beültetett rendszerek: 40 – 160 LOC / hó
- Rendszerprogramok: 150 – 400 LOC / hó
- Kereskedelmi alkalmazások: 200 – 800 LOC / hó
- Objektumpontban számolva a termelékenység 4 és 50 pont / hónap közötti, az eszköztámogatottságtól és a fejlesztők képességeitől függően.

Algoritmikus költségmodellezés COCOMO:

- Empirikus modell, a projektek gyakorlatából gyűjtött adatokon alapul.
- Jól dokumentált, hosszú tapasztalat áll mögötte. (első verzió: 1981)
- A COCOMO 2 (1995) három szintű modellt alkalmaz:
  - Korai prototípuskészítés szintje
  - Korai tervezés szintje
  - Poszt-architekturális szint

94. Hogyan értelmezhető a szoftver minősége? Milyen tényezőkkel lehet a szoftverminőséget jellemezni?

A szoftver minősége:

- A minőség általában azt jelenti, hogy a termék megfelel specifikációjának.
- Mindenki mást ért minőség alatt:
  - **A felhasználó:** „A szoftver azt végezze amit elvárok tőle, és úgy működjön ahogy én kívánom." (Ebbe beleérti a gazdaságosságot, megbízhatóságot, stb. és a ki nem mondott elvárásokat is.)
  - **A fejlesztő:** „A szoftver feleljen meg a specifikációnak"(Beleérti a karbantarthatóságot, újrafelhasználhatóságot, stb.)
- **Az ISO definíciója:** „Annak mértéke, amennyire a szoftver tulajdonságai (a minőségi jellemzők) megfelelnek a követelményeknek."
  De: mint tudjuk a szoftverkövetelmények gyakran nem teljesek és nem következetesek A szoftver specifikációját nehéz teljessé tenni, tehát a specifikációnak való megfelelés nem garantálja, hogy a felhasználó elégedett lesz a termékkel.

Tényezők:

- Biztonságosság
- Biztonság
- Megbízhatóság

- Rugalmasság
- Robusztusság
- Érthetőség
- Tesztelhetőség
- Adaptálhatóság
- Modularitás
- Komplexitás
- Hordozhatóság
- Használhatóság
- Újrafelhasználhatóság
- Hatékonyság
- Megtanulhatóság

95. Ismertesse egy szervezeten belül a minőségkezelés tevékenységeit!

- **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.

96. Miért fontosak a minőségi szabványok a szoftverkészítésben? Mi a termékszabvány és a folyamatszabvány közti különbség?

Miért fontosak?

- A szabványok adják a keretet a hatékony minőségkezeléshez.
- Lehetnek: nemzetközi-, nemzeti-, szervezeti- és projektszabványok.
- A szabványok a legjobb gyakorlat és a korábbi projektek hibáinak összegyűjtött adatai alapján készülnek.
- 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.

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

- Dokumentációs szabványok (pl. dokumentumok szerkezete)
- Kódolási szabványok (programozási stílus, programnyelv használat)

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

97. Mi az összefüggés a szoftverfolyamatok és az előállított szoftver minősége között?

- A termék minősége alapvetően függ az előállítása során alkalmazott folyamatok minőségétől (pl. iparszerű gyártásnál).
- Ez a szoftverfejlesztésnél is így van, de sok minőségi jellemző nehezen mérhető, számszerűsíthető.
- Ugyanakkor 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.

Összefüggés a szoftverfolyamatok és az előállított szoftver minősége között:
Szoftvernél ez nem ilyen egyszerű mert:

- A szoftverfejlesztésben az egyéni képzettség és gyakorlat különösen fontos.
- Külső tényezők, mint az alkalmazás újszerűsége, vagy a piacra vitel siettetése, befolyásolják a minőséget.

98. Mi a minőségi felülvizsgálat célja, milyen termékekre terjedhet ki?

Elterjedt módszer a folyamatok és a termékek minőségének ellenőrzésére. 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.
Felülvizsgálat típusai:

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

99. Mire szolgál a szoftver mérése? Mondjon néhány példát a mérhető szoftverjellemzőkre!

A szoftver mérés számszerűsíthető értékeket állít elő a szoftvertermék vagy –folyamat jellemzőiből. Célja a technikák és folyamatok objektív összehasonlítása, a minőség mérése.
Mérhető szoftverjellemzők:

- Biztonságosság
- Biztonság
- Megbízhatóság
- Rugalmasság
- Robusztusság
- Érthetőség
- Tesztelhetőség
- Adaptálhatóság
- Modularitás
- Komplexitás
- Hordozhatóság
- Használhatóság
- Újrafelhasználhatóság

100. Ismertesse a CMM (Capability Maturity Model) célját és lényegét!

A CMM - Capability Maturity Model a szervezet folyamatainak alkalmasságát méri, osztályozza és értékeli.
A CMM-modell szintjei:

- **Kezdeti:** Nincsenek hatékony vezetési eljárások, vagy hiányzik a szervezet azok következetes alkalmazására.
- **Ismételhető:** Azonos típusú projektekben ismételve a vezetési, minőségbiztosítási és változáskezelési eljárásokat sikeres lehet (a siker egyéni teljesítményektől függ).
- **Meghatározott:** A folyamatokat már definiálták, de a vezetési folyamatok még nem tudják azokat következetesen, maradéktalanul biztosítani.
- **Menedzselt:** Már vannak definiált és bevezetett folyamatok, de azok folyamatos fejlesztése még nem biztosított.
- **Optimalizált:** A folyamatok állandó fejlesztése definiált és biztosított.

101. Miért fontos a szoftver költségeinek becslése? Milyen tényezőket vehetünk figyelembe a költségbecslés során?

Becslést adhatunk arra, hogy:

- Mekkora munkát igényel egy feladat elvégzése
- Mennyi időbe kerül a feladat végrehajtása
- Mennyi a tevékenység összes költsége

Figyelembe vehetjük a:

- Piaci lehetőségeket
- A költségbecslés bizonytalanságait
- A szerződéses feltételeket
- A követelmények változékonyságát
- A fejlesztő gazdasági helyzetét

102. Ismertesse a COCOMO II. költségbecslési módszer modelljeit!

- Empirikus modell, a projektek gyakorlatából gyűjtött adatokon alapul.
- Jól dokumentált, hosszú tapasztalat áll mögötte. (első verzió: 1981)

A COCOMO 2 háromszintű modellje:

- **Korai prototípuskészítés szintje:** Becslés objektumpontok alapján.
- **Korai tervezés szintje:** Funkciópontok alapján a forráskódok számát becsli.
- **Poszt-architekturális szint:** Az architektúra terv elkészülte után becsli a szoftver méretét.

Korai prototípukészítés szintje:

- Prototípuskészítést és újrafelhasználást is figyelembe vesz.
- A fejlesztői produktivitást objektumpontokkal számolja és a CASE használatot is bekalkulálja.
- A formula:

$$PM = (NOP * (1 - \%reuse))/PROD$$

Ahol: PM – a munka emberhónapban, NOP – az objektumpontok száma, PROD - produktivitás

Korai tervezési szint:

- A követelmények tisztázása után végezhető a becslés.
- Az alábbi képlettel számol:

$$PM = A * Méret^B * M + PM_m$$

Ahol: M=PERS*RCPX*RUSE*PDIF*PREX*FCIL*SCED

$$PM_m = (ASLOC * (AT/100))/ATPROD$$

A = 2,5 a kezdeti számításban
B = 1,1 – 1,24 a projekt mérete, újdonsága függvényében.
M = projekttényezők:
PERS – személyi képességek,
RCPX – termék megbízhatóság,
RUSE – szükséges újrafelhasználás,
PDIF – platform nehézségei
PREX – személyek gyakorlata,
FCIL – támogató eszközök,
SCED – ütemezés
ASLOC = automatikusan generált kódsorok,
AT = aut. rendszerkód,
ATPRO = termelékenység,

Poszt-architekturális szint:

- Ugyanazt a formulát alkalmazza, mint a korai tervezési becslés, de két tényezőt figyelembe vesz:
  - A követelmények változékonysága,
  - A lehetséges újrafelhasználás mértéke.
- A szükséges új kódsorok számának becslésekor statisztikai és egyéb értékeket is figyelembe vesz, mint:
  - A korábbi hasonló projektek hiánya,
  - A fejlesztés rugalmassága,
  - A csapat összetartása,
  - A folyamat fejlettsége.

103. Rövidítések listája:

- **COTS:** Commercial Off-The-Shelf - Polcról levehető termékek
- **4GL:** 4th Generation Language - 4. generációs nyelvek
- **SQL:** Structured Query Language - Strukturált lekérdezőnyelv
- **BPML:** Business Process Modeling Language - Üzleti folyamatok modellezésének nyelve
- **GUI:** Graphical User Interface - Grafikus felhasználói felület
- **OO:** Obeject Orientated - Objektum orientált
- **UML:** Unified Modeling Language - Általános célú leíró nyelv
- **CORBA:** Common Object Request Broker Architecture -
- **COM:** Component Object Model - Komponens alapú objektum modell
- **DCOM:** Distributed Component Object Model - Megosztott komponens alapú objektum modell
- **API:** Application Programming Interface - Alkalmazásprogramozási interfész
- **VV**: Verification Validation - Verifikáció és validáció
- **MVC:** Model View Controller
- **CC:** Cyclomatic Complexity - ciklomatikus komplexitás
- **FP:** Function Point - Funkciópont
- **LOC:** Lines of Code - kódsorok száma
- **AVC:** nyelvfüggő szorzófaktor
- **3GL:** 3rd Generation Language - 3. generációs nyelvek
- **CMM:** Capability Maturity Model
- **COCOMO:** Constructive Cost Model

104. Források:

- Vető István 2016-os előadás és gyakorlati diái.
- 2007-es tétel kidolgozás:
https://wiki.itk.ppke.hu/twiki/pub/PPKE/Szoftvertechnol%c3%b3giaAlapjai/Vizsgatetelek07.doc
- Egyéb webes jegyzetek.