1. A szoftver-technológia funkciói

Alapfogalmak

A hardver fogalma

A hardver egyrészt a számítógépből és a hozzá csatlakozó kiszolgáló egységekből áll, másrészt pedig az ezekhez kapcsolódó fejlesztői dokumentációkból, valamint a felhasználói dokumentációkból.
A hardver összetevői:

- A számítógép
- A számítógéphez csatlakozó perifériák (monitor, nyomtató, mágneslemez)
- Kommunikációs elemek
- A rendszer felépítését, összetételét leíró dokumentáció
- A rendszer üzemeltetését, működtetését leíró dokumentáció

A szoftver fogalma

A szoftver egyrészt a számítógépen futó programokból áll, másrészt pedig a programokhoz kapcsolódó fejlesztési dokumentációkból, valamint a felhasználói dokumentációkból.
A szoftver rendszer összetevői:

- A számítógépen futó programok együttese
- A programok futtatásához szükséges konfigurációs adatok fájlok együttese.
- A rendszer felépítését, összetételét leíró rendszer-dokumentáció.
- A rendszer üzemelését, működtetését leíró felhasználói dokumentáció.

Generikus termék

Önálló termékek, amelyeket egy fejlesztő cég, intézmény saját erőből, saját kezdeményezésre, saját tervek alapján állít elő és bocsát áruba a nyílt piacon. Bármelyik felhasználó megvásárolhatja, ha szüksége van rá és képes megfizetni.
Becsomagolt szoftver. Ilyenek pl. az adatbázis-kezelő rendszerek, szövegszerkesztők, rajzoló programok, böngészők, vállalati információs rendszerek

Megrendeléses termék

A felhasználó kifejezett megrendelésére készülnek, annak az előre megszabott igényeit elégítik ki.
A fejlesztő intézmény szerződés alapján végzi el a fejlesztést a megrendelő számára.
Például: repülésirányító rendszerek, vállalatirányítási rendszerek, ipari folyamatokat vezérlő rendszerek, stb.

A hardver-technológia fogalma

A hardver rendszerek létrehozásában alkalmazott mérnöki megközelítésű tervezési, fejlesztési megoldásokat, folyamatokat, valamint a kivitelezési, gyártási folyamatokat egyesíti magában, ahol az előállított termék az elvárásoknak megfelelően működik és az előállítása hatékony módon ment végbe.

A szoftver-technológia fogalma

A szoftver rendszerek létrehozásában alkalmazott mérnöki megközelítésű tervezési, fejlesztési megoldásokat, folyamatokat, valamint a kivitelezési szoftver-előállítási folyamatokat egyesíti magában, ahol az előállított termék az elvárásoknak megfelelően működik, és az előállítása hatékony módon ment végbe.

A szoftver-technológia szempontjai

A mérnöki megközelítés értelmezése

Elméleti megfontolások, ill. kitapasztalt módszerek és bevált eszközök használata. Ha valamire nincs kidolgozott módszer, megoldás, akkor sajátot kell találni az adott problémához. A fejlesztési eredmények eléréséhez szükség van elméleti, matematikai, számítási-tudományi ismeretekre.

Az elvárásoknak megfelelő működés

A szoftver teljesíti a felhasználói igényeket elfogadható futási idő alatt, és elfogadható kényelmet biztosítva végzi el feladatát.

Hatékony előállítás

A mérnöki tevékenységek a szervezeti és pénzügyi korlátokhoz, lehetőségekhez is igazodni kell, vagyis az eredményeket a rendelkezésre álló időkereten és költségkereten belül kell elérni.
A hatékony fejlesztés: elfogadható idő alatt, elfogadható költséggel.

A szoftver-fejlesztés célja

Hatékonyság

Kellő idő alatt teljesíti a számítási feladatát. Elfogadható a futási ideje.

Megbízhatóság

Hibafellépés nélkül hajtja végre előírt feladatát.

Használhatóság

Megfelelően lehet használni, elfogadható felhasználói kényelemmel, szolgáltatásokkal.

Módosíthatóság

Könnyen meg lehet változtatni, ha a követelmények változnak.

Hordozhatóság

A lehető legkevesebb újraírás árán lehessen átvinni egy másik hardver-platformra, vagy másik operációs rendszer alá. Az a jó, ha csak újra le kell fordítani az új számítógépen. De az a legjobb, ha minden további nélkül képes üzemelni az új környezetben.

Tesztelhetőség

Könnyen lehessen tesztelni, az esetleges működési hibákat megtalálni.

Újra-felhasználhatóság

Az egész szoftver vagy annak jól elhatárolható részei más rendszerekbe is beépíthetőek, felhasználhatóak legyenek.

Karbantarthatóság:

A felhasználás során felmerülő igényekhez lehessen megváltoztatni, átalakítani, módosítani.

Együttműködhetőség

A szoftver más rendszerekkel való együttműködési, információcserélési lehetőségeire utal.

Projekt és team

A projekt

Tervezett és szervezett tevékenység egy kitűzött kutatási-fejlesztési cél megvalósítására. A projekteket ún. Team-ekbe szervezve valósítják meg.

A team

Egy projekt megvalósításában részt vevő emberek munkacsoportja. Egy projekthez több team is tartozhat, a különböző részfeladatok elvégzése céljából.

Brooks-féle szabály

Ahhoz, hogy a kezdeti kétemberes fejlesztés ne csak a két ember számára legyen elfogadható, hanem széles körben, másik által is használható legyen, még további munkára, ráfordításra van szükség, ami az eredetinek kb. 8-9-szerese.

Kezdeti prototípus-verzió vs. kereskedelmi, piacképes verzió

Különbség: A kezdeti verzió kifejlesztéséhez elegendő, ha néhány kisebb csoport végzi azt el. A piaci verzió azonban nagyon sok plusz ráfordítást igényel.
Ráfordítások: Széles körű felhasználói környezetet kell biztosítani, ami többnyire grafikus interfészt jelent. Alapos vizsgálatokra, tesztelésre van szükség a piaci kibocsátás előtt. El kell készíteni a pontos és jól kezelhető fejlesztési és felhasználói dokumentációkat. Meg kell szervezni a kereskedelmi forgalmazást és a felhasználók tájékoztatási rendszerét.

Egy szoftver-projekt legfontosabb teendői

- A feladat elemzése.
- Erőforrások felmérése.
- Projektvezetés (projekt-menedzsment) végzése.
- Követelmények meghatározása.
- A szoftver megtervezése.
- A szoftver kódolása (megírása).
- A kód tesztelése és integrálása (egybeépítése).
- A teljes rendszer működésének tesztelése.
- Üzembe helyezés (installálás).
- Dokumentálás.
- Karbantartás.
- Minőségbiztosítás.
- Felhasználók kiképzése, oktatása.

A fejlesztések sikere

Három kritériumnak való megfelelés:

- A kitűzött határidőn belül készül el.
- A fejlesztés a megállapított költségkereten belül valósul meg.
- Az eredetileg specifikált működési jellemzők többsége teljesül.

2. Szoftver alkalmazási területek

Rendszer-szoftverek

Más programok számára szolgáltatást nyújtó programok együttese. Egyes típusok fixen meghatározott információs struktúrájú fájlok feldolgozását végzik. Ilyenek pl. a fordítóprogramok, szövegszerkesztők fájlkezelők, operációs rendszerek komponensei.

Valós idejű szoftverek

Real-time software. A valós világban lejátszódó folyamatok, események időbeliségéhez igazodnak. A külső eseményeket felügyelik, elemzik, vezérlik. Fontos kritérium, hogy a szoftver reagálása, válasza minden egyes külső eseményre egy szigorúan vett időkorláton belül történjék meg. Ez azt jelenti, hogy kötött viszonyban áll az általa vezérelt külső folyamattal. Ilyen szoftverek például a vasúti forgalomirányító rendszerek.

A rendszer helyes működése függ egyrészt a rendszer által adott eredményektől, másrészt attól az időtől függ, amennyi idő alatt ez az eredmény előáll.

Megkülönböztetünk gyengén valós idejű rendszereket (soft real-time) – műveletei korlátozottak, ha a meghatározott időn belül az eredmények nem jönnek létre – és erősen valós idejű rendszereket (hard real-time) – műveletei érvénytelenek, ha az eredmények nem jönnek létre a megadott időn belül.

A valós idejű rendszerek két fő típusa:

Figyelő- és vezérlőrendszerek: időszakosan lekérdezik a rendszer környezetéről információt szerző érzékelőket. Az érzékelőkről elolvasott adatok alapján a működtetőknek utasításokat adnak.

Adatgyűjtő rendszerek: további feldolgozás és elemzés céljából gyűjtenek adatokat az érzékelőktől. A termékfolyamat egy körkörös pufferbe helyezi az adatokat, ahonnan a fogyasztófolyamat kiveszi és felhasználja azokat.

Üzleti célú szoftverek

Vállalati, üzleti információk feldolgozását végzik. Ez a legszélesebb szoftver-alkalmazási terület. Bankok, termelő, gyártó, kereskedelmi vállalatok a tipikus felhasználók, de a különböző szolgáltató cégek, valamint a közigazgatási, állami intézmények is ide sorolhatók. Főbb típusai:

Vállalati információs rendszerek

Általában egy vállalat összes tevékenységét lefedik, kiszolgálják. Óriási mennyiségű adatot dolgoznak fel és tárolnak folyamatosan. A központi adatbázist az összes komponens egységes módon tudja elérni és használni, adatokat írni bele, ill. adatokat kiolvasni. A legelterjedtebb ilyen alkalmazás az ERP System.

Vezetői információs rendszerek

Felső szintű döntéshozók, pénzügyi szakemberek és üzletemberek számára nyújtanak hatékony támogatást döntéseik előkészítése során.

Egy vállalat információs rendszerére épül és ebből egy előre kidolgozott működési modell alapján szűri ki a vezetők számára azt az információt, amire az üzleti folyamatok irányításához leginkább szükségük van.

Nem dolgozik bele a vállalati adatbázisba, onnan csak kiválasztja a hasznos adatokat.

Mérnöki és tudományos célú szoftverek

Jellemző az óriási mértékű számításigény. Mérnöki vonatkozásban ún. számítógépes tervező-rendszerek jelentik a legelterjedtebb alkalmazást.

Mérnöki célú szoftverek pl. a logikai áramkörök tervezésére, IC-k tervezésére, kártyák tervezésére, tesztelési folyamatok tervezésére használt szoftverek.

A tudományos célú szoftverek bonyolult matematikai összefüggések, képletek, differenciálegyenletek feldolgozását igénylik, amihez igen nagy teljesítményű gépek felhasználására van szükség. Pl. áramlástani problémák megoldása, űrhajók pályaszámításai, meteorológiai számítások, genetikai kutatások számításai.

Beágyazott szoftverek

A célfeladatot ellátó számítástechnikai rendszer beépül egy nem számítástechnikai rendszerbe. Valamilyen számítógépes vezérlésű eszköz, vagy nagyobb célberendezés működését segítik elő, azáltal, hogy magában az eszközben, berendezésben fejtik ki működésüket. Azokban vannak beágyazva az őket kiszolgáló CPU-val együtt.

Ilyen szoftverek pl.: mikrohullámú sütő, mosógép, mobil telefon, gépkocsiknál a gyújtásvezérlés, üzemanyag adagolás, ABS fékrendszer felügyelete, épületek fűtési, hűtési folyamatainak irányítása, gyártási folyamatok irányítása, forgalomirányító rendszerek, stb.

Személyi számítógépes szoftverek

A PC-ken felhasználható szoftvereknek rendkívül széles a választéka. Néhány fontosabb alkalmazás: Szövegszerkesztők, táblázatkezelők, adatbázis-kezelők, rajzkezelők, képszerkesztők, multimédia szoftverek, személyes üzleti alkalmazások, könyvelő programok, Levelező programok, böngészők, webhelyek, számítógépes portálok létesítésére szolgáló szoftver-eszközök.

Mesterséges intelligencia szoftverek

Olyan tevékenységeket végző informatikai rendszerek, amely tevékenységekhez intelligenciára lenne szükség, ha ember végezné. Az e téren kidolgozott szoftverek összetett feladatok megoldásában nyújtanak segítséget, nem közvetlen numerikus számításokat végezve. Jellemző rájuk, hogy az emberi gondolkodás, probléma-megoldás folyamatait próbálják meg számítógépen megvalósítani.

Pl. hangfelismerés, szövegfelismerés, alakfelismerés.

Ezeknek a szoftvereknek egy külön nagyobb kategóriáját képezik a ***szakértői rendszerek***. Ezek tudásbázissal rendelkeznek, amelybe emberi szaktudást építettek be. Működése során a rendszer ebből a bázisból válogatja ki a konkrét feladat megoldásához szükséges információt. A tudásbázis bármikor bővíthető, módosítható, az üzemeltetési tapasztalatoktól, ill. a menet közben változó követelményektől függően.

3. Fejlesztési életciklusok

Életciklus

Mindazon tevékenységek, fázisok sorozata, amelyek a fejlesztés elkezdésétől kezdve, a rendszer létrehozásán és használatba vételén keresztül addig tartanak, amíg a rendszer már végleg használaton kívül kerül.

Általános életciklus fázisok, tevékenységek

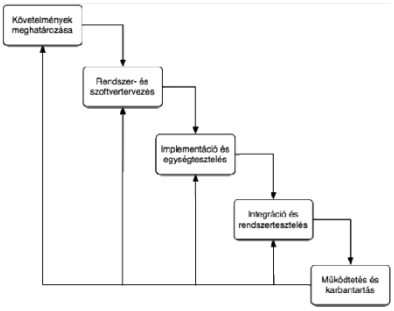
- Követelmény meghatározás
- Elemzés, tervezés
- Implementáció
- Tesztelés
- Karbantartás

Szoftverfejlesztési modellek

A különböző modellek a fejlesztési projektek egyes fontosabb aspektusainak hangsúlyozásában térnek el egymástól. Közvetlenül alkalmazhatók az OO szoftverekhez is. Fejlesztési folyamat minden egyes fázisban a fázishoz előírt feltételek betartása

Vízesés modell

A szoftver-fejlesztés első publikált modellje. Lineáris szekvenciális modell. Leginkább a vállalati információs rendszerekhez alkalmas.
Jól meghatározott követelmények és kevés változás esetén alkalmazható. Nehéz a változások figyelembevétele. Csak az előző fázis befejezése után kezdhető a következő



Követelmények elemzése és meghatározása

A szoftver rendszer szolgáltatásai, korlátai és céljai. Ezt követően az itt felsorolt tételek részletes kidolgozása. Ez adja a sw specifikációját.

Rendszertervezés és szoftvertervezés

A követelmények teljesítését megosztjuk a hardver és a szoftver között. Ezzel egy általános rendszerarchitektúra jön létre. A szoftver tervezése magában foglalja az alapvető szoftver funkciók és azok kapcsolódásának meghatározását.

Implementáció és az egységek tesztelése

Szoftver-terv programok vagy programegységek halmazaként valósul meg. Verifikáció: a programegységek tesztelése annak igazolásából áll, hogy mindegyik egység teljesíteti a specifikációt.

Integrálás és rendszertesztelés

Az egyes különálló programegységek egybeépítése, és azok tesztelése, hogy a teljes rendszerként kitűzött szoftver-követelmények teljesültek. A tesztelés után a teljes rendszer a felhasználóhoz kerül.

Üzemeltetés és karbantartás

Ez a fázis a leghosszabb az életciklusban. Rendszer üzembe helyezése és gyakorlati használatba vétele. A karbantartási szakasz ekkor veszi kezdetét. Hibák kijavítása, módosítások, átalakítások.

Evolúciós fejlesztés

Olyan rendszerek, amelyeket nem lehet vagy nem érdemes pontosan specifikálni. Több egymást követő fejlesztési állomásból tevődik össze, ahol az egyes állomások eredményeinek elemzése alapján kerül sor a következő állomás tervezésére és megvalósítására. Mindegyik állomás az előzőnél teljesebb, jobban kiérlelt verziót eredményez. A rendszer ily módon evolúciós, növekedési állomásokon megy a végső eredményig. Közvetlenül, „menet közben" igazodik a felhasználói igényekhez. Kis és közepes méretű interaktív rendszerek, nagy rendszerek részrendszereinek, rövid életciklusú szoftverek fejlesztése.

Előny: Követelményekre és a tervezésre vonatkozó döntések később.

Hátrány: A szoftver struktúrája „laza" lesz, az ilyen rendszert nehéz lesz megérteni és karbantartani. Nem jó a folyamat láthatósága. Rosszul strukturált rendszer.

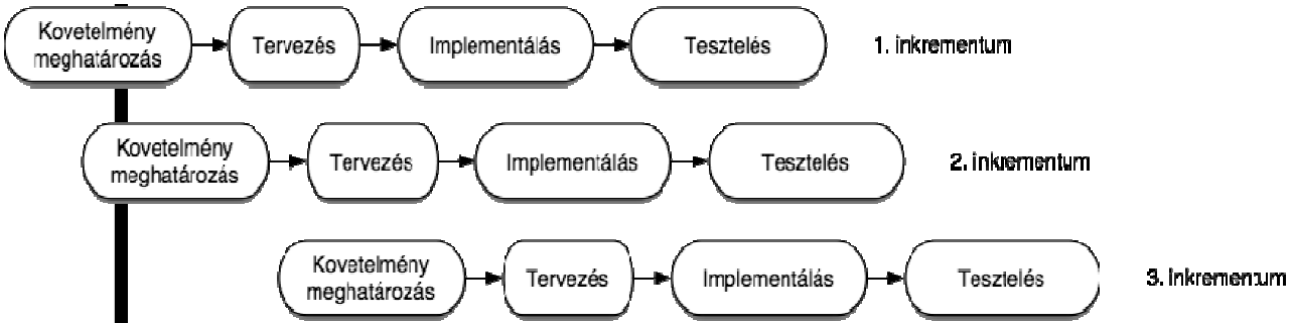
Prototípus-alapú fejlesztés

Az a fejlesztési verzió, amely alkalmas arra, hogy egy adott működési módot lehessen vele kipróbálni, vizsgálni, tanulmányozni. Ez nem tekinthető véglegesnek. Arra szolgál, hogy a megrendelő a kipróbálás után változtatni tudjon a követelményeken, aminek alapján újabb prototípus készül el az újabb vizsgálathoz. A folyamat addig tart, amíg az utolsó változat elnyeri a megrendelő elégedettségét. Az utolsó prototípus egyes részei általában beépülnek az új szoftver rendszerbe. Eldobható prototípusok: cél a követelmények megértése, tisztázása.

Inkrementális fejlesztés

A nem szigorúan specifikált kiinduláshoz alkalmazható. Növekményes modell. Követelmények prioritás alapú felosztása: a fontosabb és a kevésbé fontos szempontok a szolgáltatásokra. Fejlesztés felbontása "inkrementumokra": az egyes részfunkciókat megvalósító fejlesztési változatok.

- követelmények befagyasztása
- kívánt funkcionalitású inkrementum létrehozása
- első inkrementum - core product
- Annyi inkrementum van, ahány részre lett osztva a rendszer



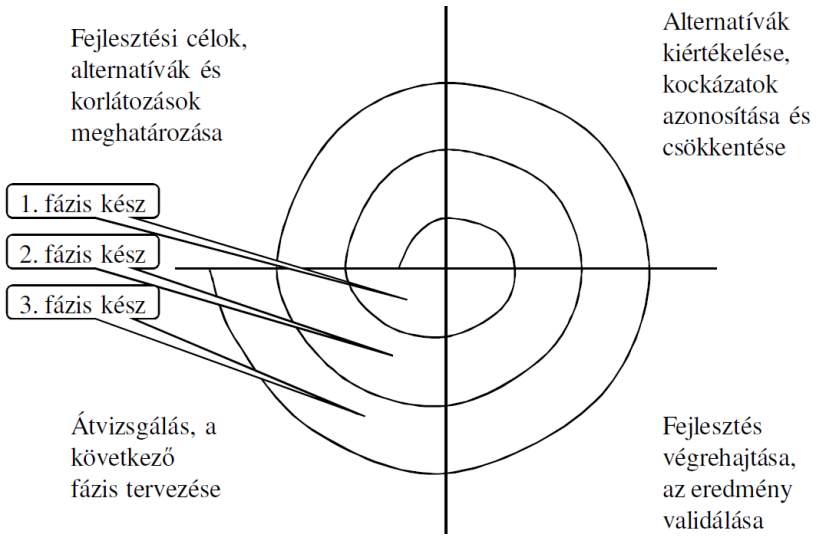
Inkrementum

Az egyes részfunkciókat megvalósító fejlesztési változatok. Egy adott inkrementum fejlesztése önállóan történik. Különböző modellek lehetségesek az egyes inkrementumoknál. Jól definiált szolgáltatásokkal rendelkező inkrementum: vízesés modell Nem világos specifikáció: evolúciós fejlesztés. Az elkészült inkrementumot hozzáintegrálják a már elfogadott és egybeépített többi inkrementumhoz. A rendszer funkcionalitása egyre inkább bővül, mindaddig, amíg el nem éri a teljes kiépítést. Jellemzői: korai megjelenés, az inkrementumok prototípusként működhetnek, kockázat csökkentése, a legfontosabb komponensek több tesztelést kapnak,

Nehézség: az egymással összefüggő inkrementumok fejlesztése.

Spirál modell

Az erőforrások elosztására és a költségekre koncentrál. Nincsenek rögzített fázisok, tevékenység szekvenciák. A szoftverfolyamatot nem tevékenységek és a köztük található esetleges visszalépések sorozataként tekinti, hanem spirálként. Minden egyes körben a spirál a szoftverfolyamat egy-egy fázisát reprezentálja (a spirál egy köre felel meg egy fázisnak) a kockázati tényezőkkel explicite számol.
A hurkok mindegyike ugyanarra a négy szektorra van felbontva:



- Fejlesztési célok, tárgykörök megállapítása
- Kockázatok értékelése és csökkentése
- Fejlesztés és validálás
- Tervezés

Fejlesztési célok, tárgykörök megállapítása

A projekt adott fázisának specifikus céljai, tárgykörei. Ide tartozik például a szoftver teljesítménye, a funkciói, stb. A projekt és a szoftver korlátozásai, és részletes menedzselési terv jön létre. Projekt kockázatok azonosítása: Kockázat: minden olyan lehetséges helyzet, ami a projekt sikeres kivitelezését akadályozhatja. Lehet műszaki, szakmai, pénzügyi, menedzselési, stb. jellegű. Az egyes kockázatoktól függően alternatív stratégiák betervezése.

Kockázatok értékelése és csökkentése

Minden egyes azonosított projekt-kockázathoz részletes elemzés készül. Intézkedések az adott kockázat csökkentésére. Például, ha fennáll a kockázata annak, hogy a követelmények nem megfelelők, egy prototípus rendszert lehet kifejleszteni, annak kipróbálása céljából.

Fejlesztés és validálás

A szoftver fejlesztési modell kiválasztása. Ha például a felhasználói interfész kockázatai dominálnak, megfelelő modell lehet az evolúciós megoldás, prototípusokkal. Abban az esetben, ha a fő kockázat az alrendszerek integrálásában rejlik, a legjobb megoldás a vízesés modell. A felhasználói követelményeknek való megfelelést a fázis befejezésekor validálással kell igazolni.

Tervezés

A projekt teljes átvizsgálása. Célja: döntés szülessen a további folytatásról. (szükség van-e egy újabb hurok megkezdésére) Ha nincsen, úgy a projekt befejeződött. Ha igen, akkor ki kell dolgozni a következő fázisra vonatkozó terveket.

V-modell

A biztonságkritikus rendszerek fejlesztésére és üzemeltetésére használják. A tervezési-ellenőrzési folyamat a baloldali ágban fentről lefelé, míg a tesztelési-ellenőrzési folyamat a jobboldali ágban lentről felfelé halad. A gyakorlatban a feladatok nem szigorúan a megadott sorrendben hajtódnak végre, valamint a tervezési fázis gyakran nagyszámú iterációt foglal magában olyan műveletek sorával, amelyeket addig kell ismételni, amíg kielégítő eredményre nem jutunk.
A fejlesztési ciklus állomásai:

Követelmények specifikálása

A funkcionális követelmények dokumentációja. A dokumentáció azt írja le, hogy milyen funkciókat és milyen módon kell a rendszernek teljesítenie

Hazárdok és kockázatok elemzése

Célja, hogy a lehetséges veszélyhelyzeteket meghatározza a megelőző kiszűrés érdekében.
Hazárd: Olyan helyzet, ami valóságos vagy lehetséges veszélyt jelent az emberekre vagy a környezetre.
Kockázat: Azt fejezi ki, hogy milyen valószínűségű veszélyes következményei lehetnek egy hazárdnak.
Az elemzési folyamatok eredményeként létrehozható a biztonsági követelmények dokumentációja. Ez a dokumentum arra ad előírást, hogy mit kell betartani a rendszernél, mit szabad és mit nem szabad megengedni a rendszer működése során

A teljes rendszer-specifikáció

A funkcionális követelmények, valamint a biztonsági követelmények együttese alkotja. Mindezen specifikáció alapján megkezdhető a teljes rendszer konkrét tervezési folyamata.

Architektúrális tervezés

Ebben a fázisban döntendő el, hogy mely funkciók legyenek megvalósítva hardver, és melyek szoftver által. Fontos szempontok lehetnek: a megvalósítási költség, a működési sebesség, a működési megbízhatóság, valamint a működési biztonságosság. A tervezés eredményeként létrejönnek a hardver és a szoftver rendszertervei, tervdokumentációk formájában.

A szoftver modulokra bontása

A tervezési folyamat egyszerűsítése, áttekinthetőbbé tétele, a feladatok részekre való bontása. A tervezés eredményeként a szoftver modulok specifikációja, valamint a köztük levő interfészek, ill. kapcsolódási folyamatok terve készül el.

Modulok előállítása és tesztelése

A modulok forráskódjának megírása, egyenkénti lefordítása, az elkészült modulok önálló tesztelése. A tesztelési folyamatok előzetesen megtervezendők. A tesztelés már integráns része a szoftver verifikációs folyamatának, amelyben azt döntjük el, hogy egy-egy modul megfelel-e a specifikációnak.

Rendszer integrálása és tesztelése

A gyakorlatban kétféle megközelítés terjedt el erre:
Inkrementális tesztelés: egyenkénti bővítés és tesztelés. Előnye, hogy az új hibák egy-egy bővítés során léphetnek be nagy valószínűséggel, s így könnyebb felfedni őket.
Big bang tesztelés: az összes modult egyszerre rakjuk össze és egy menetben teszteljük. Ez a megoldás azon a feltevésen alapul, hogy az előzetesen kitesztelt modulok együtt is helyesen fognak működni. Ekkor a hibák megtalálása viszonylag nehezebb lesz a feladat bonyolultsága miatt.

A teljes rendszer verifikálása

Ebben a fázisban eldöntendő, hogy a teljes rendszer funkcionálisan teljesíti-e az összes specifikációs pontot. Ehhez felhasználandó a rendszer terve, valamint a teljes rendszer-specifikáció,

A teljes rendszer validálása

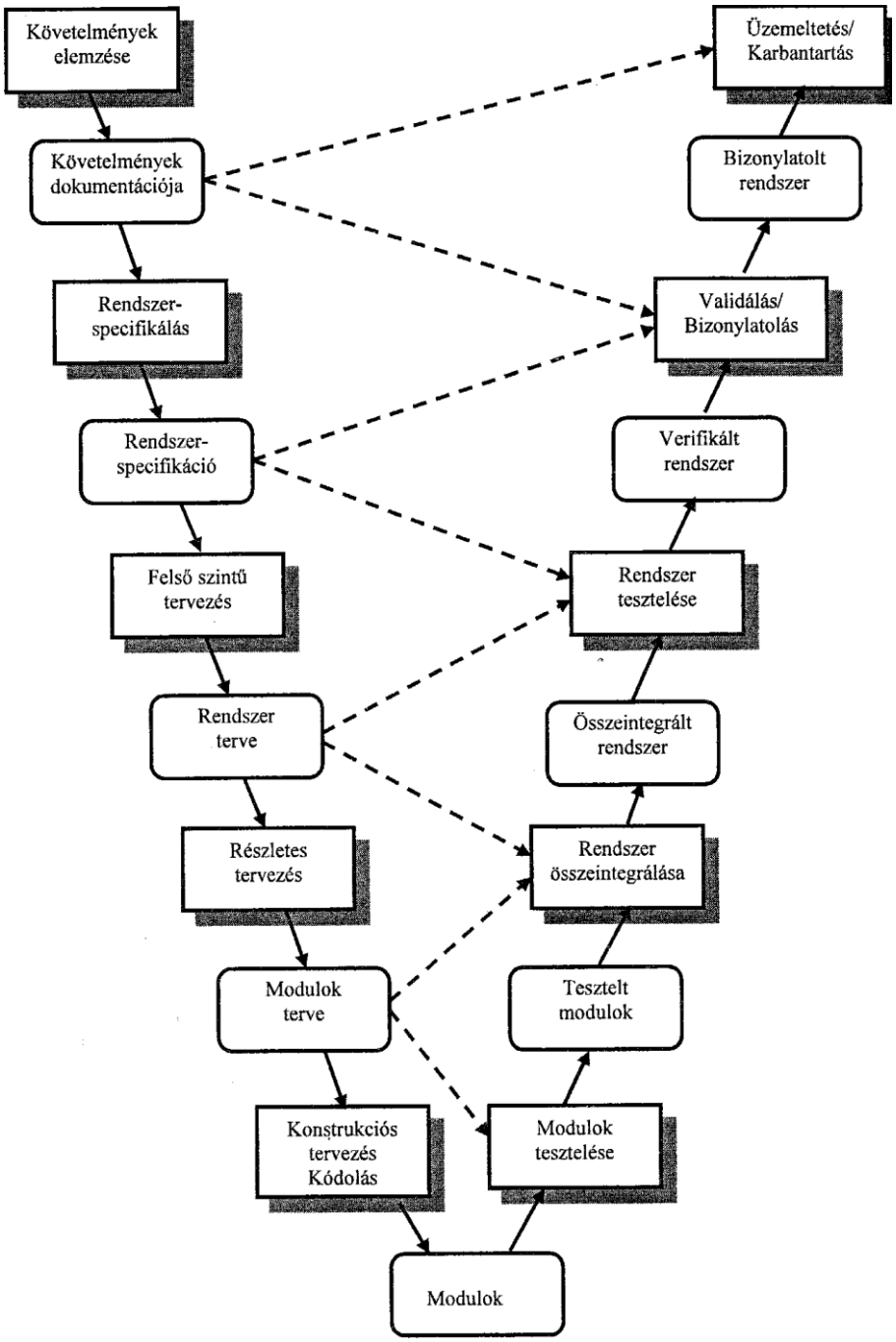
Ebben a fázisban eldöntendő, hogy a teljes rendszer megfelel-e a felhasználói követelményeknek. Ebbe beletartozik a biztonsági feltételek teljesítésének eldöntése is, az ún. biztonságigazolás.

Bizonylatolás

A hatósági előírások és szabványok szerinti megfelelés eldöntése, igazolása, és az erre vonatkozó bizonylatok kiállítása.

Rendszer üzemeltetése, karbantartása

Üzembe helyezés, üzemeltetés, karbantartás, elavulás, üzemeltetés megszüntetése.



4. Biztonságkritikus rendszerek

Alapfogalmak

Biztonságkritikus rendszer

Olyan informatikai rendszer, amely azzal az elsődleges követelménnyel működtetendő, hogy ne veszélyeztesse az emberi életet, egészséget, ne okozzon gazdasági vagy környezeti károkat. Pl. vasúti állomásirányító központok, atomerőművek vezérlése, űrrepülők irányítása, kórházakban, bankokban, hadászatban alkalmazott információs rendszerek.

Hibatűrő rendszer

Egy informatikai rendszer hibatűrő, ha képes elvégezni, ill. tovább folytatni előírt feladatának helyes végrehajtását hardver- és szoftver-hibák esetén. Ez nem azt jelenti, hogy a rendszer ugyanazzal a hatásfokkal, teljesítménnyel fog működni hibásan is. Teljesítménye lecsökkenhet, esetlegesen nem mindegyik funkcióját látja el maradéktalanul viszont a legfontosabb feladatait végre tudja hajtani.

Minőségi és megbízhatósági követelmények

Megbízhatóság

Annak feltételes valószínűsége, hogy egy informatikai rendszer hibátlanul működik a $[t_0, t]$ időintervallumban feltéve, hogy a $t_0 < t$ időpillanatban hibátlanul működött. A valószínűségi függvény jele: R(t). Azt fejezi ki, hogy a rendszer a $[t_0, t]$ időtartamban végig hibátlanul működik. Ez az időtartam hosszának növekedésével csökken. Ezt a paramétert olyan rendszerek jellemzésére használják kiemelten, ahol egy rövid időre sincs megengedve a hibás működés vagy nincs mód javításra.

Rendelkezésre állás

Annak valószínűsége, hogy az informatikai rendszer helyesen működik a t időpontban, vagyis a t időpontban rendelkezésre áll. A valószínűségi függvény jele: A(t). Az A(t) mérőszám értéke ott lehet fontos, ahol a rendszer nem állandóan használják egy feladatra, hanem gyakori megszakításokkal. Ilyen esetekben megnövekszik a közbenső karbantartás és javítás szerepe, amely a rendelkezésre állás magas szinten való tartását szolgálja.

Biztonságosság

Annak valószínűsége, hogy az informatikai rendszer helyesen vagy hibásan működik a t időpontban, de oly módon, hogy nem veszélyeztet emberi életet, nem okoz anyagi vagy környezeti kárt, és nem befolyásolja károsan más rendszerek működését. A valószínűségi függvény jele: S(t). Ha a rendszerben komoly meghibásodás lép fel, a biztonság fenntartása úgy oldandó meg, hogy a rendszer fokozatosan állítja le egyes funkcióit, ezzel ellensúlyozva a hardver- és szoftver-hibákat. Ezt nevezzük fokozatos leépülésnek. A másik megoldás hogy a rendszer, még utolsó intézkedéseként úgy áll le, hogy a katasztrófa semmiképpen ne következhessen be.

Karbantarthatóság

Annak valószínűsége, hogy a meghibásodott rendszer újra működőképessé tehető t időtartam alatt. Ebbe az időbe beletartozik a hiba helyének meghatározása, a fizikai javítás vagy csere elvégzése, valamint az újraindítás. A valószínűségi függvény jele: M(t). Az M(t) értékének magas szinten tartása feltétlenül megköveteli a beépített öntesztelést, valamint az automatizált diagnózis elvégzését, amely a hiba meghatározására irányul.

Verifikáció és validáció

Verifikáció

Az a folyamat, amelyben meghatározzuk, hogy a vizsgált informatikai rendszer szoftvere teljesíti-e mindazokat a követelményeket, amelyeket egy előző fázisban specifikáltak a szoftverfejlesztési vagy előállítási folyamatában.

Validáció

A teljes szoftver vizsgálata és kiértékelése azzal a céllal, hogy meghatározzuk, minden szempontból megfelel-e a felhasználó követelményeinek.

5. Fejlesztési projektek

Mérföldkő

A mérföldkő egy kiemelt teljesítési időpont a projekt menetében. Másképpen: egy kiemelt teljesítési állomás a projekt menetében. Ez lehet egy fontos fejlesztési fázis befejezése, egy nagyobb szoftver-egység előállítása, egy tesztelési folyamat lezárása, egy dokumentáció elkészítése, stb.

A kritikus út

Az egymás után elvégezendő tevékenységek azon sorozata, amely meghatározza a projekt teljes időtartamát.

Tartalékidő

Tartalék idővel csak a nem kritikus tevékenységek rendelkezhetnek, a kritikus úton pedig nincsen semmilyen időtartalék.

6. A modulokra bontás elve

Alapfogalmak

Egy modul által hívott modulok számát **szétágazás**nak, egy modulhoz érkező hívások számát **összeágazás**nak nevezzük.

Hívási lánc

A hívásban egymást közvetlenül követő modulok sorozata, amelyben a legelőször szereplő modul az út kezdő csomópontja, a legutolsó pedig a befejező csomópontja

Hívási lánc hossza

A hívásban egymást közvetlenül követő modulok száma.

Hívási szint

A főprogramtól lefelé láncban kapcsolódó modulok maximális száma.

Hívási szélesség

Az ugyanazon a szinthez tartozó modulok maximális száma. Másképpen a hívási szétterjedés maximális mértéke.

7. Mérőszámok alkalmazása

Funkció-orientált számok

A funkciópont

Egy adott funkciópont úgy számítható ki, hogy egy működési részterülethez tartozó szoftverfunkciók számát szorozzuk a részterülethez rendelt súlyértékkel. Ezek a részterületek a következők:

Felhasználói bemenetek száma

Ennyi különböző vezérlési mód áll a rendelkezésünkre.

Felhasználói kimenetek száma

Ennyi különböző futási eredmény áll a rendelkezésünkre. Ebbe az informatív üzenetek, hibaüzenetek is beletartoznak.

Felhasználói lekérdezések száma

Azok a lekérdezések tartoznak ide, amelyek on-line módon történnek, és azonnali válasz érkezik rájuk.

A kezelt fájlok száma

A szoftver által előállított, vagy feldolgozott különböző fájlok tartoznak ide.

Külső interfészek száma

Egy másik, külső rendszerhez való csatlakozás interfészeinek száma.

Mérőszámok a forráskódra

A Halstead-számok

A paraméterek kifejezetten egy modul forráskódjára vonatkoznak. A teljes szoftver mérőszámai a modulokra vonatkozó számok összegeként állítható elő. A Halstead-féle mérőszámok a program kódoláselméleti méretére és bonyolultságára nyújtanak összehasonlítási alapot

A négy legfontosabb mérőszám:

n1: a programban található különböző műveletek, ill. műveleti jelek száma.
n2: a programban található különböző operandusok száma.
N1: az összes előforduló műveletek, ill. műveleti jelek száma a programban.
N2: az összes előforduló operandusok száma a programban.

A forráskód becsült hossza kódoláselméleti megfontolásból:

$$N(b) = n1 * \log_2 n1 + n2 * \log_2 n2$$

A forráskód aktuális hossza a kódoláselmélet üzenethosszának elve szerint:

$$N = N1 + N2$$

A program tárgykódjának számított bitmérete (volumene):

$$V = N * \log_2 (n1+n2)$$

A McCabe-féle szám

A strukturális komplexitásra McCabe vezetett be mérőszámot. A McCabe-szám (MC) a programban lévő logikai döntések számától függ, mégpedig a következő módon:

$$MC = D+1$$

Ahol D a logikai döntések száma.

McCabe szerint, ha ez a szám 10 fölött van, akkor már nehezen áttekinthető, nehezen kezelhető, nehezen tesztelhető programot kapunk. A 20 fölötti érték pedig már gyakorlatilag kezelhetetlen modult jelent. Ezért arra kell törekedni, hogy egy programmodulban lehetőleg ne legyen 10-nél több elágazásunk. Inkább bontsuk az adott modult két vagy több részre.

Architektúrális mérőszámok

A különböző szoftverek általános strukturális összehasonlítására szolgálnak.

Card és Glass mérőszámai

Strukturális komplexitás

$$S(i) = fan\text{-}out(i)^2$$

Adat-komplexitás

$$D(i) = \frac{v(i)}{fan\text{-}out(i)+1}$$

Ahol v(i) az i-edik modulhoz tartozó input és output változók száma együttesen.

Modul komplexitás

$$C(i) = S(i) + D(i)$$

Architektúrális komplexitás

$AC = \sum_{i=1}^{M} C(i)$, a*hol M a modulok száma.*

Az AC érték növekedésével várhatóan növekedni fog az integrálási és tesztelési ráfordítás mértéke. Az AC és a hibaszám között nagyjából lineáris az összefüggés

Henry és Kafura mérőszámai

Henry és Kafura olyan mérőszámot javasolt, amely magában foglalja az összeágazások számát is.
Ők a következő komplexitási mérőszámot definiálták:

$$HKM(i) = L(i) * [fan\text{-}in(i) + fan\text{-}out(i)]^2$$

Ahol L(i) az i-edik modul forráskódjának hossza.

Mindezek után az M modulból álló teljes szoftver komplexitása:

$$HKM = \sum_{i=1}^{M} HKM(i)$$

8. Fejlesztési ráfordítások

A mérőszámok összefüggésben vannak az általuk jellemzett szoftverek fejlesztési ráfordításaival.

Egy szoftver-projekt tervének alapjában véve három összetevője van:

- A fejlesztés célja. Az elvégzendő munka.
- A rendelkezésre álló erőforrások, amelyekkel a projekt elvégezhető.
- A rendelkezésre álló pénzügyi források, amelyek a projektre fordíthatók.

Az erőforrások legfontosabb elemei:

- Az emberek, akik a fejlesztést végzik.
- A meglevő hardver eszközök, és a felhasználó szoftver eszközök (OS, CASE eszköz, toolok)

Rayleigh-függvény

A szoftver-fejlesztésben a legfontosabb erőforrás az emberi oldalon jelentkezik. Ez a leglényegesebb összetevő, és egyúttal a legtöbb költség is erre megy el.
Fontos, hogy a team-ekben legyenek fejlesztési szakértők kellő szaktudással, kreativitással. Ezekből a kulcsemberekből elég csak néhánynak lenni. Ha egy ilyen sem lenne, a projekt eleve kudarcba fulladna.
Másik szempont a fejlesztők létszáma. A fázisok teljesítéséhez különböző létszámú emberi erőforrás felhasználására lehet szükség. A projekt átfutási ideje alatt folyamatosan változik a szükséges résztvevők száma.
A nagyméretű projektekre érvényesül az ún. **Rayleigh-eloszlás**, ami az emberi ráfordításnak az időtől való függését írja le. A függvény: **RC (t)** (Resource Consumption).

$$RC(t) = \frac{t}{k^2} \cdot e^{-\frac{t^2}{2k^2}}, ahol\ 0 \leq t < \infty$$

k egy konstans, ami az RC függvénynek azt a helyét adja, ahol az a maximális értékét veszi fel. A **k** az adott projektre jellemző érték.
A ráfordítás mértéke eleinte alacsony, utána gyorsan felnövekszik a csúcsig, majd ezt követően csökken, de már lassabban, mint ahogyan növekedett.
Nem érdemes az összes meghatározó szakembert a projekt egészében alkalmazni.

A szoftverköltség becslése

Szoftver rendszer fejlesztési összetevői:

- Hardver és szoftver költségek – nagyjából állandónak tekinthető egy projektre nézve
- Utazás, kiképzés költségei – szintén állandó mértékű a projekt egészére nézve
- Emberi ráfordítások – domináns tétel

Az eddigi tapasztalatok azt mutatják, hogy egy szoftver-mérnök fizetése felett, egy fejlesztőre nézve még kb. ugyanannyi költség esik a kiegészítő személyzet és a munkahelyi infrastruktúra fenntartására.

A legfontosabb költségbecslés **algoritmikus úton** végezhető el.

A résztvevők számának növekedésével csaknem négyzetesen emelkedik a páronkénti kommunikáció ideje. A modulok számának növekedésével exponenciálisan megnövekszik az integrálási és a tesztelési költség.

A legáltalánosabb formájában a következő **becslési összefüggést** szokás alkalmazni:

$$Ráfordítás = A * W * Méret^B$$

Az **A** egy konstans tényező, amelynek értéke függ a szoftver jellegétől, típusától, bonyolultságától.
A **W** a projekt tulajdonságaitól függő tényező.
A **Méret** a szoftverre vonatkozik.
A **B** kitevő értéke általában 1 és 1.5 közé eső szám, a projekt bonyolultságától függően.

A legjobban használható becslési alap a kódsorok száma.

Három értékre kell kiszámolni a költség képletét:

- legrosszabb eset kódmérete
- várható kódméret
- legjobb eset kódmérete

A kódméret becslésére fel lehet használni a funkciópont-értéket. Ez úgy végezhető, hogy korábbi projektek adatelemzése révén meghatározzuk, hogy átlagosan mennyi kódsor kellett egy funkciópont megvalósításához, egy adott programozási nyelven. Legyen ez a szám **AVC (avarage number of lines of code)**.

*Kódméret = AVC * funkciópontok száma*

Sommerville szerint az AVC értéke:

- assembler nyelv esetén 200-300 LOC/FP
- egy modern magas szintű nyelv esetén 20-40 LOC/FP

A fejlesztő intézmény részéről két lényeges szempontból lehet szükség a ráfordítás minél megbízhatóbb előrejelzésére.

1. saját fejlesztés várható költségeinek ismerete
2. A megrendelésre elvállalt fejlesztés árának eldöntéséhez.
   Ár = Saját költség + Profit

Az emberi ráfordítás mérése

Emberhónap

(EH, ehó)
A létszámot szorozzuk az időtartammal.
Azt a tényt, hogy X ember Y hónapig dolgozik, az X→Y kapcsolattal jelöljük.

Legyen például egy projekt menete a következő:
4E→3H, 15E→6H, 6E→3H
Tehát a projekt időtartama 3H + 6H + 3H = 12H

A projektben a teljes időtartam alatt részt vevő emberek átlagos létszámát a következő módon kapjuk:

$$LSZ = \frac{4E \cdot 3H + 15E \cdot 6H + 6E \cdot 3H}{3H \ + \ 6H \ + \ 3H} = 10E$$

9. A szoftver tesztelése

Definíciók

Bemeneti adat

Olyan számítógépes adat, amely egy szoftver működtetését vonja maga után, felhasználói szinten.

Kimeneti adat

Olyan számítógépes adat, amely egy szoftver működtetése során jelenik meg, a működtetés eredményeként felhasználói szinten.

Hibamodell

Azon szoftver-hibák halmaza, amelyeknek a felfedésére irányul a teszttervezés.

Tesztelés

A szoftver bemeneti adatokkal való sorozatos ellátása és a kimeneti válaszadatok megfigyelése.

Egy hiba tesztje

Bemeneti adatok olyan rendezett sorozata, melynek hatására az adott hibát tartalmazó szoftver a legutolsó adatkombinációra a hibás kimeneti adatkombinációval válaszol. Ebben az esetben azt mondjuk, hogy a teszt kimutatja, felfedi, vagy detektálja a hibát. Itt használatban van még az a kifejezés is, hogy a teszt lefedi az adott hibát.

Halmaz-leképezési modell

A hibás működés szemléltetésére alkalmazzák. A modellben egy adott program az összes lehetséges bemeneti adatból előállítja az összes lehetséges kimeneti adatot.
Bevezetett jelölések:

INPUT – Az összes lehetséges bemeneti adat halmaza.
OUTPUT – Az összes lehetséges kimeneti adat halmaza.
I_H – A hibás működést okozó bemeneti adatok halmaza.
O_H – A hibás kimeneti adatok halmaza.

Teszt: egy bemenő adatcsoport akkor és csak akkor lesz a szoftverben meglévő h_i hiba tesztje ha benne van az I_H halmazban.

Detektálható hiba

Egy hiba akkor detektálható, ha legalább egy tesztje létezik. Tervezési **redundancia** folytán létezhetnek olyan hibák, amelyek semmilyen körülmények között nem éreztetik hatásukat a szoftver egészének működésében. Az ilyen hibák **nem detektálhatók** a teszt definíciója értelmében.

Tesztkészlet

Több hiba tesztjének együttese, halmaza.

Tesztkészlet hibalefedése (fault coverage)

Azon hibák százalékos aránya az összes feltételezett hibához képest, amelyeket a tesztkészlet detektálni tud. Azon hibák százalékos aránya az összes előre feltételezett hibához képest, amelyeknek van tesztjük a tesztkészletben. Az elérhető cél a 100%-os lefedést produkáló tesztkészlet létrehozása.

Teljes tesztkészlet/teszthalmaz

Teszteknek olyan összessége, amely a szoftver mindegyik előre feltételezett és detektálható hibájának tesztjét magában foglalja. Az ilyen teszthalmazról azt mondjuk , hogy teljes hibalefedést eredményez. A teljes teszthalmaz 100%os hibalefedésű.

Teszttervezés

A bemeneti tesztsorozat és az elvárt válaszjelek meghatározása egy előre feltételezett hibahalmazra nézve.

Tesztszámítás vagy determinisztikus tesztgenerálás

Egy adott hiba tesztjének szisztematikus számításokkal történő meghatározása.

Véletlenszerű/random tesztgenerálás

Bemenő adatok sorozatának véletlenszerű képzése a hibák tesztjeként történő felhasználásra. Ilyenkor a bemeneti tesztadatokat egy adott tartományban használt véletlenszám-generátorral állítjuk elő.

Hibamodellek és érvényességi körük

Hibamodell

Azoknak az előre feltételezett hibáknak a konkrét halmaza, amelyekre vonatkozóan a teszttervezést végre kívánjuk hajtani. Egy hibamodell főbb meghatározási szempontjai:
A teszttervezés kivitelezhetőségi lehetőségei és a ráfordítandó költségek.
Az adott fejlesztési technológiához kapcsolódó hibajelenségek előzetes ismerete.
A tesztelés végrehajtásához szükséges hardver és szoftver eszközök által nyújtott szolgáltatások köre.

Szoftver-hibák

A szoftver rendszerek hibás működése két okra vezethető vissza:

Szoftver-specifikációs hibák

- fejlesztés kezdetekor megjelenő hibák – nem teljesíti a felhasználói követelményeket.
- téves specifikáció – működésében nem azt írták elő, amire valójában szükség lett volna
- ellentmondásos specifikáció – két vagy több előírt működési mód ellentmondásban van.
- hiányos specifikáció – egy adott felhasználási helyzetre nem írták elő az elvárt működést.

Programozási hibák

A szoftver tervezése és kódolása során a programozó által elkövetett hibák körét foglalja magában.

- Hibás funkcióteljesítés,
- hiányzó funkciók,
- adatkezelési hibák az adatbázisban,
- indítási és leállítási hibák,
- felhasználói interfész hibái,
- határértékek túllépése,
- kódolási hiba,
- algoritmikus hiba,
- kalkulációs hibák,
- inicializálási hiba,
- vezérlési folyamat hibája adattovábbítási hiba,
- versenyhelyzet programblokkok között

Funkcionális tesztelés (fekete doboz tesztelés)

A programot egyedül csak a külső viselkedésében, az előírt funkcióinak teljesítésében vizsgáljuk. A bemeneti adatokra kapott kimeneti válaszadatokat figyeljük. Egyáltalán nem vesszük figyelembe a forráskódot. A programot olyan egységnek tekintjük, amibe nem látunk bele. A szoftver minden egyes specifikált funkciójának a sorra vétele és leellenőrzése a cél, ezen kívül pedig még az is, hogy nincs-e valamilyen téves, nem betervezett tevékenysége a programnak.

Strukturális tesztelés (fehér doboz tesztelés)

A szoftver kódjának felhasználásával a belső struktúra, a belső működés végigkövetésére irányuló ellenőrzés. Az a cél, hogy a forráskód utasításainak és elágazásainak minél alaposabb végigjárását tudjuk elérni. Itt a programot tehát olyan egységnek tekintjük, amibe belelátunk. A strukturális tesztelés a program minden egyes utasításának a végrehajtását, a döntési elágazások lehetséges végrehajtásait, valamint a ciklusok végrehajtását kell ellenőrizni.

A programtesztelés pszichológiája és gazdaságossága (Glenford Myers elvei)

Tesztelés célja

A program olyan végrehajtása, amelyben a szándékunk a hibák megtalálása.
A megbízhatóság növelése azáltal érhető el, hogy megkeressük és eltávolítjuk a hibákat.
Nem lehetséges egy programot úgy tesztelni, hogy garantálni tudjuk annak hibamentes voltát.
A tesztelés alapvető szempontja a gazdaságosság, vagyis annak elérése, hogy minél több hibát fedjünk fel egy véges számú vizsgálattal.

Tesztelési elvek

- Egy tesztnek szükséges része az elvárt kimenet vagy eredmény definiálása.
- Célszerű a tesztelést a fejlesztéstől független személy vagy szervezet által elvégeztetni.
- A teszteket nemcsak elvárt és érvényes bemeneti feltételekre kell írni, hanem olyan feltételekre is, amiket nem várunk el, ill. amik érvénytelenek.
- Annak a valószínűsége, hogy egy programszakaszban még hibák léteznek, egyenesen arányos az ugyanott már megtalált hibák számával

A teszttervezés kulcskérdése

Hatékony tesztek megtervezése. Nem lehetséges a meglevő összes hiba tesztjének előállítása. A legkisebb hatékonyságú a véletlenszerű tesztelés. Előtérbe helyezzük a determinisztikus úton történő tervezést, ahol arra törekszünk, hogy minél átgondoltabban alakítsuk ki a felhasználásra kerülő tesztkészleteket. Ajánlatos a fejlesztési fázisokat az itt leírt sorrendben végezni: előbb a fekete azután a fehér.

Az integrációs tesztelés

Szoftver modulok összeépítése, integrálása alatt végrehajtandó tesztelési folyamatok.

A tesztelés akkor is elengedhetetlen, amikor az egymáshoz kapcsolt modulokat előzőleg már minden tekintetben levizsgálták. Arról van szó ugyanis, hogy az együttes működés még külön alapos ellenőrzést követel meg, mivel nem biztos, hogy a modulok az interfészeiken keresztül helyesen kommunikálnak, és jól látják el feladataikat.
Az integrációs vizsgálatokat azokban az esetekben is el kell végezni, amikor kipróbált és korábban már bevált szoftver-komponenseket használunk fel egy új fejlesztési környezetben.

Modulok tesztelése és integrálása

Modulon értjük azt a szoftver-egységet, amely önálló névvel van ellátva, önállóan azonosítható és hívható-futtatható kódrész tartozik hozzá, saját, megkülönböztetett változónevekkel.

Mielőtt egybeépítenénk a modulokat, mindenképpen el kell végezni az elkülönített, önálló vizsgálatukat. A tesztelés a modul működési specifikációja alapján történik, amelyben a funkciók megadása szerepel, valamint a különböző bemeneti adatokra elvárt kimeneti adatok leírása.

A funkciók és az interfészek ellenőrzése fekete dobozos megközelítést kíván. Ezzel szemben a modultesztelés nagymértékben fehér dobozos kategóriájú.

A modulok ellenőrzése egy olyan verifikációs folyamatnak felel meg, amelyben a modultervek és az elkészült programok közötti ekvivalencia igazolását végezzük el.

Az integrálás szervezése igen fontos kérdés. A két legelterjedtebb megoldás a következő:

Együttes tesztelés (big bang testing)

A levizsgált modulok mindegyikét összeépítjük és együttesen, egyszerre kezelve őket indítjuk el az ellenőrzést, vagyis a teljes programot futtatjuk.

Inkrementális tesztelés

Lépésenként, többnyire egyenként bővítjük a már levizsgált modulok halmazát az újonnan beépítendő modullal. A folyamatban minden egyes új összeépítettség-állapotban újonnan teszteljük.

- hívási hierarchia szerint – lentről felfelé vagy fentről lefelé
- hívási szálak szerint – az egymás után sorra vett modulhívások láncszerűen végigkövetve

Egy modul izolált ellenőrzése

Az önállóan lefordítható és a tárgykódjukkal eltárolt szoftver-komponensek együttműködésének megteremetésére az operációs rendszerek külön szervező programjai, az ún. Linkage Editorok szolgálnak. Ezek a kötési kapcsolatok meglétét ellenőrzik, másrészt a futás közben történő vezérlés-átadást valósítják mg. Egy modul akkor és csak akkor tud futni, ha az összes kapcsolata létre van hozva, a teljes szoftver pedig csak akkor indulhat el, ha mindegyik modulja be van már linkelve.
Egy a környezetéből kiragadott modul izolált ellenőrzése csak úgy mehet végbe, ha megteremtjük számára azt a környezetet, ami a memóriába való betöltését és futási végrehajtását teszi lehetővé.

Meghajtó modul

Olyan főprogram, amely a következő célokat szolgálja: meghívja a tesztelt modult, ellátja a szükséges vezérlő paraméterekkel, a tőle kapott számítási adatokat kiírja.

Csonk modul

A tesztelt modul által kiadott hívás szimulálására szolgál. Annyi ilyenre van szükség, ahány különböző modul kerül meghívásra. Egy csonk modul annál hasznosabb, minél inkább teljesíti az általa helyettesített igazi modul funkcióit.

Megjegyzések

Az adott modultól függően érdemes lehet külön meghajtót írni a fekete és fehér dobozos ellenőrzésre.

Az együttes tesztelés végrehajtása

Csak akkor hajtható végre, mikor már mindegyik modul le lett ellenőrizve. A végrehajtás során igen nagy valószínűsége van annak, hogy a teljes rendszer működési hibákat fog mutatni.
Az együttes tesztelés legnagyobb hátránya a hibalokalizálás nehéz kivitelezhetősége. Az együttes tesztelést akkor érdemes alkalmazni ha:

- a rendszer működése jól áttekinthető
- a modulok világos és tiszta funkcionális elkülönüléssel rendelkeznek
- az izolált tesztelésük az interfészekre is alaposan kiterjedt
- a fejlesztés magas minőséggel, a hibaelkerülési elvek messzemenő betartásával ment végbe.

Az inkrementális tesztelés menete

Először a két modul egybekapcsolódásával indul. Ha ezek működése megfelelő, akkor a harmadik modul bekapcsolása és az így kialakult csoport ellenőrzése következik, ezt követően a negyedik modul jön, és így tovább amíg az utolsó modul is beépül a halmazba.
Jelöljük a modulokat nagybetűvel, az integrálási állapotokat halmazokkal, a hozzájuk rendelt teszteket pedig indexelt T betűvel, abban a sorrendben, ahogy végrehajtásra kerülnek. Egy állapot után csak akkor lépünk tovább, ha már kijavítottuk az összes ott feltárt hibát.
A bővítéseknél megismételjük az összes korábbi tesztet az újabbal kiegészítve. Erre azért volt szükség, hogy ellenőrizzük nem zavart-e bele a korábban levizsgált csoport működésébe az új modul. Ezt regresszív vagy regressziós tesztelésnek, visszaható tesztelésnek nevezzük.

Hierarchikus végrehajtás

A procedurális szoftverekre jellemző a modulok közötti hívási hierarchia. A legelőször elinduló főprogram által hívott alprogramok további hívásokat adnak ki, amelyek fentről lefelé terjednek. Kivételt képez a rekurzív kapcsolódás.

Fentről lefelé haladó integrálás

A tesztelésben ilyenkor fel lehet használni a már beépült modulokat is. A csonk modulok megírására és felhasználására lesz szükségünk, a meghajtó modulok szerepét maguk a már letesztelt eredeti szoftverelemek fogják ellátni.

Lentről felfelé haladó integrálás

Csak a meghajtó modulok megírására van szükség, a csonk modulok szerepét maguk a már letesztelt komponensek látják el.

Összehasonlítások

A fentről lefelé történő haladás általában több nehézséggel jár. Akkor előnyösebb, ha a nagyobb, ill. komolyabb hibák fent jelentkeznek a hierarchiában.
Az építkezési irányokat kombinálni is lehet. Ilyenkor a két irányból való haladás eredményeként az integrálás a megfelelő moduloknál össze fog érni. Ezt **szendvicstesztelésnek** nevezzük.
A modulok közötti interfészek hibái korábban kiderülnek az inkrementális tesztelésben. A modulok menet közben látják egymást, nem pedig a legvégén.
Ha egy hibát észlelünk az inkrementálisnál, ez vagy az új modulban lesz várható vagy az új modul egyik kölcsönhatásában. Ezzel szemben az együttes tesztelésnél ezt a hibát nehéz lenne lokalizálni.
Az együttes teszteléseknél a különböző modulokat több fejlesztő tudja egyidejűleg vizsgálni.

10. Komponens-alapú fejlesztések

CORBA – szabványosított rendszer, ami a különböző fejlesztésű szoftver-komponensek közötti együttműködést teszi lehetővé. Átgondolt szisztematikus újrafelhasználás.

Az irányzat kialakulása

A komponens-alapú fejlesztés egy olyan folyamat, amelyben egymástól független komponenseket tervezünk be és integrálunk egy közös szoftver rendszerbe. Egyre inkább terjed, ennek okai:

- A szoftver rendszerek egyre összetettebbé, bonyolultabbá válnak, a jelentkező igények miatt.
- A vevők, megrendelők minél gyorsabban akarnak kész szoftver-termékhez jutni.
- A szoftver-terméknek megbízhatónak kell lennie, a gyors fejlesztés mellett is.

A nagy komplexitás, a rövidebb átfutási idő, és a nagy megbízhatóság együttesen csak úgy érhető el, ha kipróbált, kész, jól bevált szoftver-komponenseket használunk fel, ahelyett hogy újonnan fejlesztenénk ki őket.

Főbb jellemzők

1. független, önálló komponenseket használ fel, amelyek az interfészükkel vannak specifikálva. Az interfész és a belső megvalósítás egymástól el vannak választva. Ez azt jelenti, hogy a belső megvalósítás le is cserélhető egy másikkal, de az interfész működése változatlan marad.
2. A komponensek szabványosítva vannak, ami elősegíti az integrálásukat. A szabványok előírják, hogy miként kell specifikálni a komponensek interfészeit, és hogyan kell kommunikálni a komponenseknek. Teljesen mindegy, milyen nyelven írják. Különböző nyelven fejlesztett elemek ugyanabban a szoftverben integrálhatóak.
3. **Közvetítő szoftver (middleware)** – az egymástól független elemek együtt tudjanak működni, kezeli az egymás közti kommunikációt. pl. CORBA. Egy jó middleware a biztonságot is növeli.
4. Olyan tervezési és fejlesztési folyamatra van szükség, ami kifejezetten a komponens-alapú technológiához van igazítva.
5. A komponensek nem zavarnak bele egymás működésébe. Mivel a belső működésük rejtve van, ezért bármikor újraírhatóak és kicserélhetőek, anélkül, hogy felborítanák a teljes rendszer működését. Ehhez az szükséges, hogy az interfész változatlan maradjon.

Problémák, nehézségek

Egy komponens megbízhatósága – Ha vásárolt elemről van szó, a felhasználó nem lehet 100%-ig biztos abban, hogy hibátlan és pontosan úgy működik, ahogy le van írva mivel nincs forráskód.

A komponens bizonylatolása – Kívánatos lenne, hogy egy piacra vitt komponenst egy független értékelő szervezet vizsgáljon meg és hagyjon jóvá. Vagyis adjon rá egy bizonylatot. Még az sincs eldöntve, hogy ki fizesse a vizsgálatok elvégzését.

A komponens velejáró tulajdonságai – Teljesítmény, hibatűrés, megbízhatóság. Elemről kiderül, hogy olyan tulajdonságai vannak, amelyek nem jó irányban befolyásolják a teljes rendszer működését.

Kompromisszum a követelményekben – Az ideális követelményeket a valósághoz kell igazítani. A valóságot a rendelkezésre álló komponensek jelentik. A tervező intuíciójára vagyunk utalva.

Alapvető tulajdonságok

- függetlennek kell lennie
- valamilyen komolyabb, alapvető funkciót kell megvalósítania
- felhasználható legyen rendszerépítésre, a generikus fejlesztőtől különböző másik cégnél.
- ***Szabványosított*** - Valamilyen szabványos modellnek kell hogy megfeleljen a komponens.
- ***Független*** – Önmagában legyen felhasználható. Ne kelljen hozzá kiegészítő komponens.
- ***Beépíthető –*** Az összes kapcsolódása definiálva legyen, az interfészek pontos megadása révén.
- ***Telepíthető*** – Önálló egységként lehessen egy számítógépen futtatni. A kód bináris formában áll a rendelkezésre és fordítás nélkül futtatható
- ***Dokumentált*** – A felhasználásával kapcsolatos összes információt tartalmaznia kell a dokumentációnak. Ebben benne van az interfész részletes specifikációja is. A potenciális felhasználó ennek alapján tudja eldönteni azt, hogy az adott komponens megfelel-e az ő igényeinek.

Interfész típusok

Szolgáltatási interfész – Azokat a szolgáltatásokat definiálja, amiket a komponens nyújt a többi komponens felé. Lényegében ez a komponens API-ja. Jelölés: egy kör a vonal végén.

Igénylési interfész – Azokat a szolgáltatásokat definiálja, amiket a komponens fogad és használ a környezetéből. Ezeket a többi rendszerelemnek kell biztosítania. A komponens független és önálló, de az adott feltételek mellett érvényesül. Jelölése: egy nyitott félkör.

Fejlesztési kérdések

- azonnal elindíthatóak
- nem definiálnak adattípust, egy komponensről nem származik át sehová semmi
- A megvalósításuk ismeretlen lehet, a felhasználó előtt
- nyelvtől függetlenek
- szabványosítva vannak

Komponens-modellek

Milyen szabványok legyenek a komponens megvalósítására, dokumentálására és beépítésére nézve. Ezek a szabványok a komponensek fejlesztőinek szólnak. A közvetítő szoftverek kötelesek igazodni a komponensek működéséhez. A komponens-modellek meglehetősen bonyolultak. Főbb elemeik:

- **Interfészek**
  - az interfész leírási módja – miként kell definiálni a komponens interfészét. Mik legyenek benne a leírásban. Milyen formális nyelvet kell használni a leírásban.
- **Felhasználási dokumentáció**
  - Névhasználat – a hálózati alkalmazásban a CORBA az Internet domain-neveinek a használatához igazodik
  - Meta-adatok – a komponens saját adatai, amelyeket a felhasználónak ismernie kell a komponens működtetése során
  - Konfigurálás – az adott futtatási környezethez való igazítás
- **Telepítési információ**
  - dokumentálás – azt írja elő, hogy mi legyen a dokumentációban.
  - telepítés – azt írja elő, hogyan kell a komponenst előkészíteni ahhoz, hogy futtatható állapotban legyen a felhasználó számára.

Használati kérdések

Egy komponens fejlesztésénél igen fontos mérnöki feladat megtalálni az optimális kompromisszumot a bonyolultság és a használati kényelem között. A komponens-használat egyik fontos területe a vállalatok, cégek által birtokolt, régóta meglévő, régóta használatban lévő szoftver rendszerek világa.
Nehéz és költséges lenne őket egy újra cserélni ezért érdemes lehet ezeket egy új rendszer komponenseként felhasználni mivel bevált, kipróbált megoldásokat hordoznak magukban. Az egyetlen lehetséges megoldás az, hogy az eredeti komponenst teljesen körbeveszik olyan interfésszel, amely kapcsolatot teremt a régi szoftver és az újak között. Ezt **csomagoló programnak (wrapper program)** nevezik. A csomagoló program feladata az, hogy a régi szoftver összes funkcióját közvetítse a külvilág felé.

A komponens-alapú fejlesztés folyamata

Kész elemekből építkezik. A megfelelő modellben a fejlesztés a következő fázisokra bontható:

1. A rendszerkövetelmények meghatározása.
2. A számításba vehető komponensek azonosítása.
3. A követelmények módosítása a megtalált komponensekhez igazodva.
4. Architektúrális tervezés
5. A számításba vehető komponensek azonosítása.
6. A komponensek egybeépítése, a teljes szoftver rendszer létrehozása.

Vannak lényeges különbségek, amik fennállnak a komponens-alapú fejlesztés és a nulláról történő fejlesztés között. Ezek a következők:

1. A felhasználói követelményeket elég csak körvonalazni.
2. A követelményeket finomítani és módosítani kell már a folyamat elején
3. Miután megtörtént az architektúrális tervezés újabb komponens-keresés beiktatásra van szükség. Ekkor kiderülhet az, hogy bizonyos kiválasztott komponensek nem működnek megfelelően a többivel együtt. Vissza kellhet menni az 1-es fázishoz, vagy pedig a 3-as fázishoz.
4. A folyamat befejezése. Integráljuk a számításba vett komponenseket. Ahol két vagy több komponens között nincs meg a szükséges információs kapcsolat, ott interfész programok megírására van szükség. A hiányzó komponenseket önállóan ki kell fejleszteni és integrálni.

A komponensek azonosítása

Három résztevékenységre bontható:

1. Komponens keresése
2. Komponens kiválasztása
3. Komponens validálása

Azonosításra két fázisban (2, 5) van szükség. A 2. fázisban azt kell eldöntenünk, hogy vannak-e a céljainknak megfelelő komponensek. Miután a rendszer-architektúra meg lett tervezve, már részletes validációs tesztelést kell végrehajtani a komponenseken. Ekkor arról kell meggyőződni, hogy az azonosított komponens valóban beleillik-e az alkalmazásba.
A validációs eljárás mélysége - Ha ismert és megbízható forrásból, akkor egyedileg nem is kell tesztelni, elég lesz majd csak akkor tesztelni, amikor a többi komponenssel össze van integrálva. Ha viszont ismeretlen forrásból való komponens, akkor egyedileg is mindig le kell ellenőrizni a működését.
A validálás abból áll, hogy előzetesen megtervezzük a végrehajtandó tesztek sorozatát. Azt, hogy milyen bemenő adatokat küldünk a komponensre, és milyen kimenő válaszadatokat várunk el.
Ha egy komponens több funkcióval rendelkezik, meg kell vizsgálni, hogy a számunkra fölösleges funkciók nem okoznak-e zavarokat a saját felhasználásunkban.

11. A karbantartási folyamatok

A verifikációs és validációs folyamatok elvégzése után kerül sor a szoftver rendszer kibocsátására, a normál üzemeltetés megkezdésére.
Bármilyen jól sikerült is a szoftver, előbb-utóbb szükségessé válik, hogy valamilyen változtatást hajtsanak végre rajta. Változhatnak a felhasználói igények, vagy pedig szükségessé válik egy új hardver platform vagy operációs rendszer bevezetése.
Az üzembe helyezést követően, a normál felhasználás alatt elvégezendő szoftver-változtatások folyamatát szoftver-karbantartásnak nevezzük. Ez kis beavatkozástól lényeges átdolgozásig terjedhet. Ez is része a szoftver életciklusának.

Karbantartási típusok

Javító karbantartás

A felhasználás során kiütköző hibák szükségszerű kijavítása. Különböző mértékű befektetés:

- kódolási hiba – kis ráfordítás
- tervezési hiba – nagyobb ráfordítás
- specifikációs hiba – legnagyobb ráfordítás (rendszer újratervezéssel is járhat)

Adaptív karbantartás

A szoftver megváltoztatása annak érdekében, hogy egy másik hardver platformon, vagy egy másik OS alatt lehessen tovább üzemeltetni. A szoftver funkcionalitása lényegében változatlan marad.

Kibővítő karbantartás

Az üzemelés alatt jelentkező felhasználói és piaci igények kielégítése érdekében változásokat, bővítéseket hajtanak végre. Új funkciókat is visznek be, amivel bővül a szoftver felhasználási skálája.

A ráfordítás aránya

A karbantartási mérőszám: SMI

A változtatások mértékének kifejezésére vezették be a szoftver érettségi fokát, az SMI-t. Az adott szoftver stabilitásának, változatlanságának a mértékét fejezi ki.

M = a jelenlegi kibocsátású szoftverben lévő modulok száma
F(vált) = megváltoztatott modulok száma
F(új) = újonnan hozzáadott modulok száma
F(tör) = törölt modulok száma

$$SMI = \frac{M - [F(vált) + F(új) + F(tör)]}{M}$$

Ha nincs változtatás, akkor az SMI=1 lesz. Ha változtatás történik SMI <1 lesz. Ha a karbantartási verziók SMI-számai közelítenek az 1-es érték felé, akkor a szoftver-termék kezd stabilizálódni, egyre kevesebbet változik. Az SMI számértékei összefüggésben állnak a karbantartás ráfordítási költségeivel.

Heckenast – Kogníció, koncepciók modellek

Skálatípusok

Nominális skála

Bizonyos kategóriához való hozzátartozás kifejezésére. Pl.: entitások címkézése, osztályozása

Ordinális skála

Elemek közötti sorrend fejezhető ki. A sorrend valamilyen attribútum szerint alakul ki.
Pl.: preferencia, keménység, levegő minősége, intelligencia tesztek

Intervallum skála

A mért értéket ekvivalens intervallumok számával fejezzük ki. Pl.: „relatív” idő (naptári), hőmérséklet.

Arány skála

Kifejez sorrendet. Pl: idő intervallum, hosszúság: szoftverkód hossza, fizikai objektumok hossz

Abszolút skála

A mérés egy bizonyos entitás elemeinek megszámlálását jelenti. Egyetlen leképezés lehetséges, az aktuálisan számolt szám. A mérés eredményével mindenfajta aritmetikai analízis értelmes.
Pl.: egy adott személy életkora, egy adott program forrássorainak száma