23. Szoftver-tesztelés

Kérdések

- Mi a különbség a validációs tesztelés és a hibatesztelés között?
- Mik a rendszer- és komponenstesztelés alapelvei?
- Milyen stratégiákat alkalmazhatunk tesztgenerálás céljára?
- Mik az automatizált tesztelés eszközeinek alapvető jellemzői?

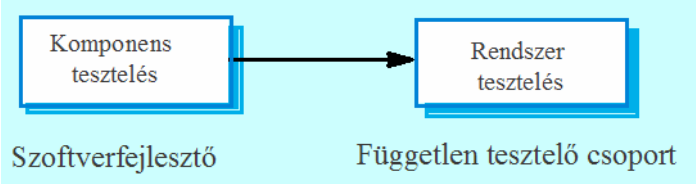
Tartalom

- Rendszertesztelés
- Komponens tesztelés
- Tesztgenerálás
- Automatikus tesztelés

A tesztelési eljárás

- Komponens tesztelés
  - Programkomponensek egyedi tesztelése;
  - Általában a komponens fejlesztőjének feladata (kivétel a kritikus rendszerek);
  - A tesztek a fejlesztő tapasztalatán alapulnak.
- Rendszertesztelés
  - Komponensek rendszerbe vagy alrendszerbe integrált csoportjainak tesztelése;
  - Egy független fejlesztő csoport feladata;
  - A tesztek a rendszerspecifikáción alapulnak.

A tesztelés fázisai



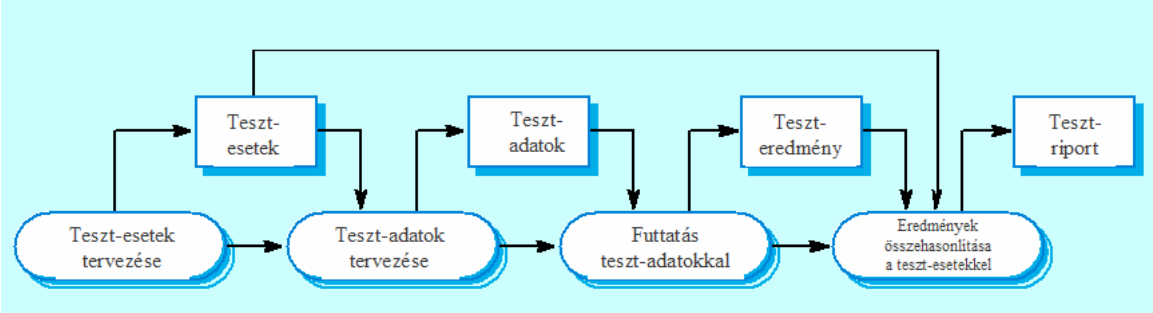
Hibatesztelés

- A hibatesztelés célja a programhibák felfedezése
- A *sikeres* hibateszt a programot hibás viselkedésre bírja
- A teszt csak a hibák jelenlétét mutatja, de nem képes a hibák hiányát jelezni.

A tesztelési eljárások célja

- Validációs tesztelés
  - A fejlesztők és megrendelő számára annak demonstrálása, hogy a szoftver teljesíti a követelményeket;
  - A sikeres tesz eredménye, hogy a rendszer a tervek szerint működik.
- Hibatesztelés
  - A rendszerben hibák keresése: hibás viselkedés, vagy a specifikáció nem teljesítése;
  - A sikeres teszt a rendszert hibás működésre kényszeríti, így a rendszer hibájára mutat rá.

A szoftvertesztelés folyamata



Tesztelési vezérelvek

- Csak a kimerítő teszteléssel deríthető ki, hogy a program hibamentes. De a kimerítő tesztelés lehetetlen.
- A tesztelési vezérelvek definiálják a tesztek kiválasztásának módját:
  - A menükön keresztül elérhető valamennyi funkciót le kell tesztelni;
  - Az azonos menün keresztül elérhető funkciók kombinációit tesztelni kell;
  - Ahol felhasználói bevitel van, minden funkciót ellenőrizni kell helyes és helytelen adatokkal.

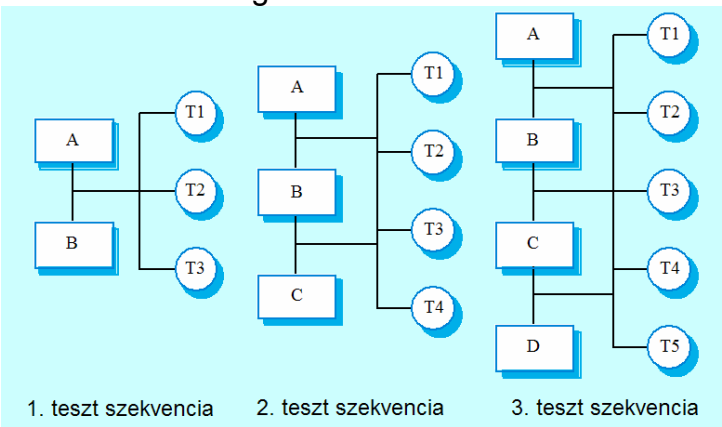
Rendszertesztelés

- A komponensek rendszerbe vagy alrendszerbe integrálásával foglalkozik.
- A megrendelőnek átadandó inkrementum tesztelésével is foglalkozhat.
- Két fázis:
  - Integrációs teszt – A tesztelők használhatják a rendszer forráskódját. A rendszer a komponensek integrálása folyamán teszteljük.
  - Végteszt (*release test*) – A tesztelők az átadandó teljes rendszert fekete dobozként tesztelik.

Integrációs tesztelés

- A rendszer komponensekből áll. A tesztelés a komponensek interakciójából eredő problémákkal foglalkozik.
- Felülről lefelé (*top-down*) integrálás
  - A rendszer vázának felépítése, majd a komponensek beépítése.
- Alulról felfelé (*bottom-up*) integrálás
  - Az infrastruktúra komponensek integrálása, majd a funkcionális komponensek hozzáadása.
- A hibalokalizálás megkönnyítése érdekében a rendszereket inkrementálisan célszerű integrálni.

Inkrementális integrálás-tesztelés



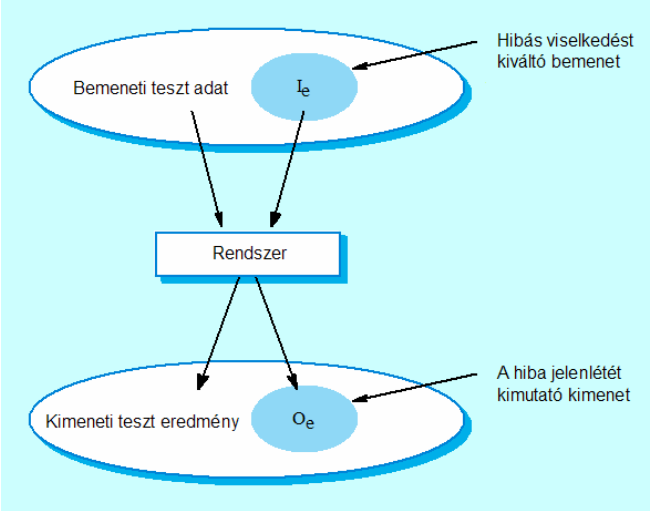
Tesztelési megfontolások

- Achitektúrális validáció
  - A felülről lefelé integrálásos teszteléssel könnyebb a rendszer architektúrájában levő hibák felfedezése.
- Rendszer demonstrációja
  - A felülről lefelé integrálásos tesztelés a fejlesztés korai szakaszában már lehetővé tesz korlátozott demonstrációt.
- A teszt implementálása
  - Gyakran könnyebb alulról felfelé integrálásos teszteléssel.
- A teszt megfigyelése
  - Mindkét megközelítéssel problematikus. Extra kódra lehet szükség a teszt megfigyeléséhez.

Végtesztelés

- A rendszer egy terjesztésre szánt verziójának (*release*) tesztelésének folyamata.
- Elsődleges célja a gyártó bizalmi szintjének növelése abban, hogy a rendszer megfelel a követelményeknek.
- A végteszt általában fekete doboz teszt (vagy funkcionális teszt)
  - Csak a specifikáción alapul;
  - A tesztelők nem ismerik a rendszer implementációját.

Fekete doboz tesztelés



Tesztelési tanácsok

- Ötletek a tesztelőknek: hogyan érdemes olyan teszteket választani, amelyek kimutatják a rendszer hibáit.
  - Olyan bemenetek választása, amelyek a rendszert hibaüzenetek (az összes!) generálására kényszerítik;
  - Olyan bemenetek tervezése, ami puffer túlcsorduláshoz vezethet;
  - Ugyanazon bemenet vagy bemeneti sorozat többszöri ismétlése;
  - Érvénytelen kimenetek kikényszerítése;
  - Túl nagy vagy túl kicsi számítási eredmények kikényszerítése.

Szcenárió-alapú tesztelés

Egy diák Amerikai történelmet tanul és éppen dolgozatot ír a polgárháborúról. Ehhez forrásokat keres különféle könyvtárakban. A LIBSYS rendszerbe bejelentkezve a kereső szolgáltatást használja eredeti dokumentumok keresésére. Talál is néhány forrást amerikai egyetemek könyvtáraiban és le is tölt onnan néhány másolatot. Az egyik dokumentumhoz azonban igazolásra van szüksége az egyetemétől, hogy valóban hallgató és a letöltés nem szolgál kereskedelmi célokat. Az igazolás kiállítását a LIBSYS rendszeren keresztül kéri. Ha az igazolást megkapja, akkor a dokumentumot letöltik a könyvtár szerverére és kinyomtatják. A diák egy e-mail üzenetet fog kapni, amelyben értesítik, hogy átveheti a dokumentumot.

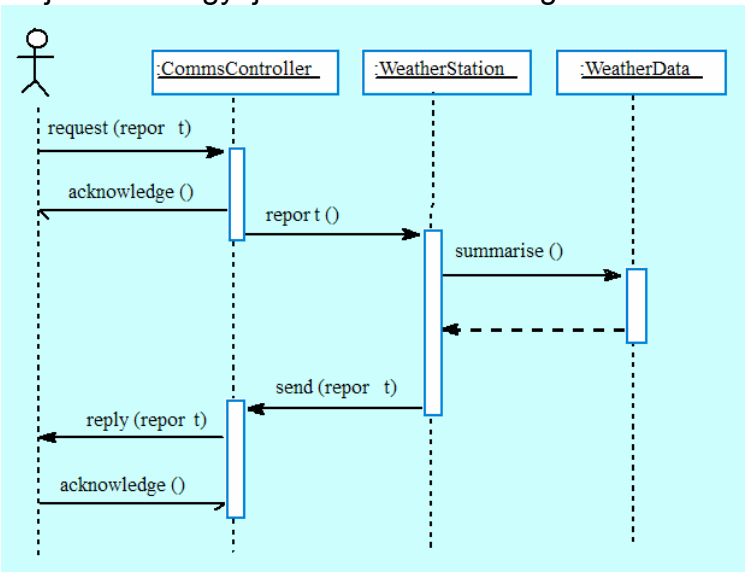
Rendszertesztek

- A bejelentkezési mechanizmus tesztelése helyes és helytelen azonosítókkal: annak ellenőrzése, hogy az érvényes azonosítókat elfogadja, az érvényteleneket visszautasítja.
- A keresés tesztelése különféle kereső kifejezésekkel ismert forrásokra: ellenőrizhető, hogy a kereső valóban megtalálja-e a dokumentumokat.
- A kijelzés tesztelése: a dokumentumokról szóló információ helyesen van-e kijelezve?
- A letöltéshez engedélyt kérő mechanizmus tesztelése.
- Az e-mail-es értesítő rendszer tesztelése: elküldi-e a levelet a dokumentum megérkezésekor.

Használati esetek

- Használati esetek (use cases) alapján tesztek készíthetők. Segítenek a tesztelendő operációk kiválasztásánál és a szükséges teszt esetek megtervezésében.
- A kapcsolódó szekvencia-diagramból a teszt számára a bemenetek és kimenetek meghatározhatók.

Időjárási adat gyűjtése szekvencia diagram



A teljesítmény tesztelése

- A végteszt egy része a rendszer eredő tulajdonságainak tesztelése, pl. teljesítmény, megbízhatóság.
- A teljesítmény tesztelése általában egy teszt-sorozattal történik, ahol a terhelést fokozatosan növeljük, amíg a rendszer teljesítménye már elfogadhatatlanná válik.

Stressz tesztelés

- A rendszert a tervezett értéknél jobban terheljük. A rendszer stresszelése gyakran fed fel hibákat.
- A stresszelés a hibás működés közbeni viselkedését is teszteli. A rendszernek nem szabad katasztrofálisan összeomlania. Teszteli az elfogadhatatlan szolgáltatás-kiesést vagy adatvesztést.
- A stressz teszt különösen fontos elosztott rendszereknél, ahol a rendszer súlyosan degradálódhat, ha a hálózat túlterhelődik.

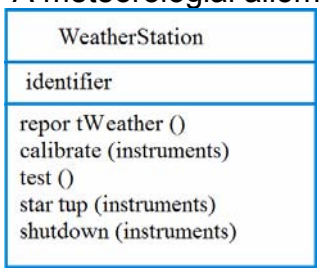
Komponens tesztelés

- A komponens tesztelés az egyes komponensek izolált tesztelésének folyamata.
- *Hibatesztelő* folyamat.
- Komponensek lehetnek:
  - Egyedi függvények vagy objektumok metódusai;
  - Objektum osztályok sok attribútummal és metódussal;
  - Kompozit komponensek, amelyek szolgáltatásait interfészeken keresztül lehet elérni.

Objektum osztályok tesztelése

- Egy osztály teljes tesztelése:
  - Az objektumhoz tartozó valamennyi operáció tesztelése;
  - Az összes attribútum beállítása és lekérdezése;
  - Az objektum valamennyi állapotának elérése.
- Az öröklés megnehezíti az objektum-osztály tesztelését, mert a tesztelendő információ nem lokalizált.

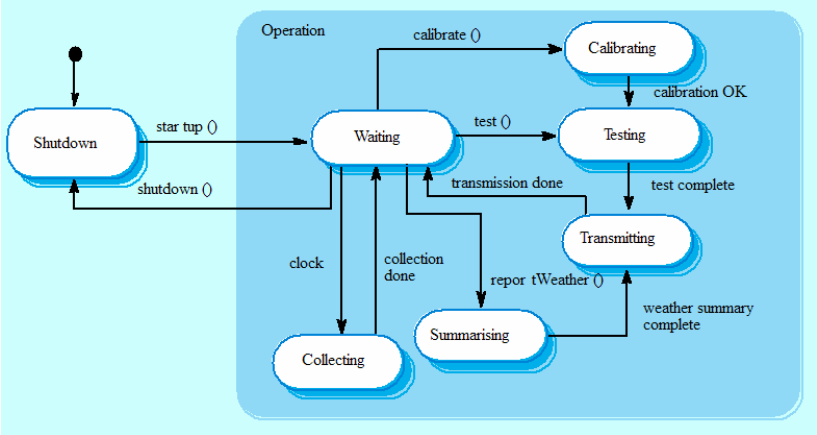
A meteorológiai állomás objektum interfésze



A meteorológiai állomás tesztelése

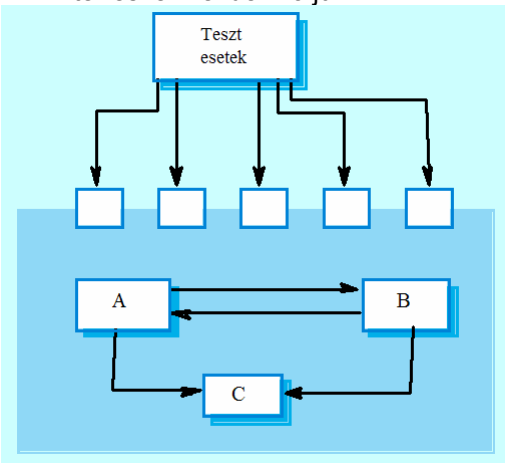
- Teszt eseteket kell definiálni a metódusokhoz: reportWeather, calibrate, test, startup, shutdown.
- Az állapotgép modell segítségével a tesztelendő állapotváltozások sorozatának kijelölése, majd az ezekhez tartozó esemény-sorozat meghatározása.
- Példa:
  - Waiting -> Calibrating -> Testing -> Transmitting -> Waiting

A meteorológiai állomás állapot-diagramja



Interfész-tesztelés

- A cél az interfészek hibáinak, vagy az interfészekről alkotott hibás feltételezésekből eredő hibák felderítése.
- Nagyon fontos objektum-orientált fejlesztés esetén, amikor is az objektumokat interfészeikkel definiáljuk.



Interfészek típusai

- Paraméter interfészek
  - Adat átadása egyik eljárásból a másikba.
- Osztott memória interfészek
  - Egy közös memóriarészt használ több eljárás vagy függvény.
- Procedurális interfészek
  - Egy alrendszer eljárásokat tartalmaz, amelyeket más alrendszerek hívhatnak.
- Üzenettovábbításos interfészek
  - Komponensek más komponensektől üzeneteken keresztül szolgáltatást kérnek.

Interfész hibák

- Hibás interfész használat
  - A hívó komponens egy másik komponenst akar használni, de rosszul használja annak interfészét (pl. rossz paraméter-sorrend).
- Interfész félreértelmezés
  - A hívó komponens a hívott komponens viselkedéséről téves feltételezésekkel él.
- Időzítési hibák
  - A hívó és hívott komponensek más sebességgel működnek és így előfordulhat elavult adatok használata.

Interfész tesztelési tanácsok

- A tesztet úgy kell megtervezni, hogy a hívott eljárás paraméterei a tartomány határán legyenek.
- A mutató típusú paramétereket mindig tesztelni kell null-pointerekkel is.
- A komponens hibás működését előidéző tesztek tervezése.
- Üzenettovábbítással működő rendszerek stressz-tesztelése.
- Osztott memóriás rendszerekben az aktivált komponensek sorrendjének változtatása.

A teszt esetek tervezése

- A tesztelés során használt teszt esetek (bemenetek és kimentek) tervezésével foglalkozik.
- A teszt esetek tervezésének célja hatékony tesztek készítése validációs és hibatesztelés céljára.
- Tervezési módszerek:
  - Követelmény-alapú tesztelés;
  - Partíciós tesztelés;
  - Strukturális tesztelés.

Követelmény-alapú tesztelés

- A követelmény-tervezés egyik alapelve, hogy a követelmények tesztelhetők legyenek.
- A követelmény-alapú tesztelés egy *validációs* tesztelési technika, ahol minden egyes követelményhez kidolgozunk azt ellenőrző teszteket.

LIBSYS követelmények

- A felhasználó a teljes adatbázisban kereshet, vagy kiválaszthatja ennek egy részhalmazát.
- A rendszer biztosítja a tárolt dokumentumok megfelelő megjelenítését.
- Minden megrendelés egyéni azonosító (ORDER_ID) alapján letölthető az előfizető tárhelyére.

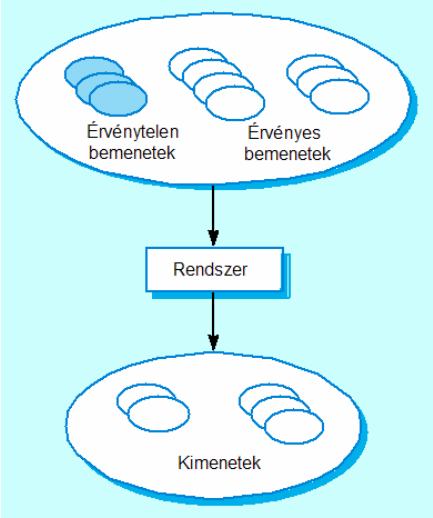
LIBSYS tesztek

- Keresés olyan elemekre, amelyekről tudjuk, hogy jelen vannak, valamint olyanokra, amelyek nincsenek. Az elérhető adatbázisok száma legyen 1.
- Keresés olyan elemekre, amelyekről tudjuk, hogy jelen vannak, valamint olyanokra, amelyek nincsenek. Az elérhető adatbázisok száma legyen 2.
- Keresés olyan elemekre, amelyekről tudjuk, hogy jelen vannak, valamint olyanokra, amelyek nincsenek. Az elérhető adatbázisok száma legyen több, mint 2.
- Az elérhető adatbázisokból egy kiválasztása, abban keresés olyan elemekre, amelyekről tudjuk, hogy jelen vannak, valamint olyanokra, amelyek nincsenek.
- Az elérhető adatbázisokból több kiválasztása, azokban keresés olyan elemekre, amelyekről tudjuk, hogy jelen vannak, valamint olyanokra, amelyek nincsenek.

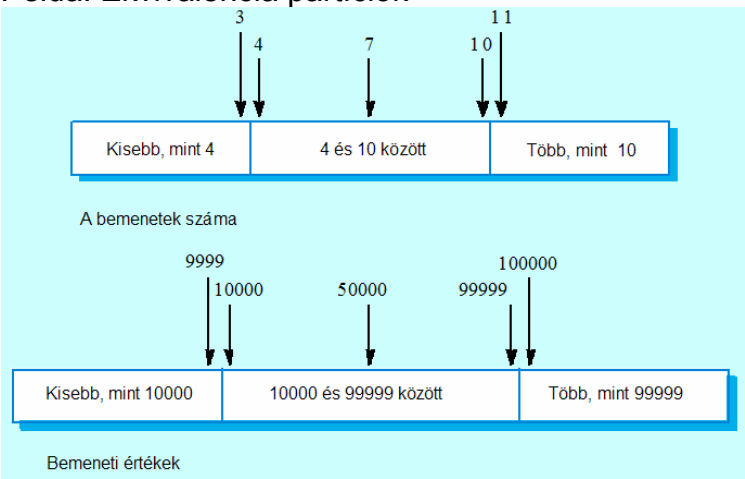
Partíciós tesztelés

- A ki- és bemeneti adatok gyakran különböző osztályokba sorolhatók, ahol az osztályon belüli elemek „hasonlóak”.
- Ezen osztályok mindegyike egy ekvivalencia partíció, ahol a a program ekvivalens módon viselkedik az osztály minden elemére.
- Teszt esetek minden partícióból választandók.

Ekvivalencia partíciók



Példa: Ekvivalencia partíciók



Példa: Keresési algoritmus specifikációja

```
procedure Search (Key : ELEM ; T: SEQ of ELEM;
       Found : in out BOOLEAN; L: in out ELEM_INDEX) ;

Pre-condition
      -- a sorozatnak legalább egy eleme van
      T'FIRST <= T'LAST
Post-condition
      -- a keresett elemet megtaláltuk  az L-ik helyen
      ( Found and T (L) = Key)
or
      -- az keresett elem nincs a tömbben
      ( not Found and
            not (exists i, T'FIRST >= i <= T'LAST, T (i) = Key ))
```

Keresési algoritmus – bemeneti partíciók

- Olyan bemenetek, amelyekre igaz az előfeltétel (pre-condition).
- Olyan bemenetek, amelyekre nem igaz az előfeltétel.
- Olyan bemenetek, amelyekre a keresett elem megtalálható a tömbben.
- Olyan bemenetek, amelyekre a keresett elem nincs a tömbben.

Tesztelési tanácsok (sorozatok)

- Szoftver tesztelése olyan sorozattal, aminek egy eleme van.
- A különféle tesztekben más és más méretű sorozatok használata
- A tesztekben az első, utolsó és középső elem felhasználása.
- Tesztelés olyan sorozattal, aminek nulla eleme van.

Keresési algoritmus – bemeneti partíciók

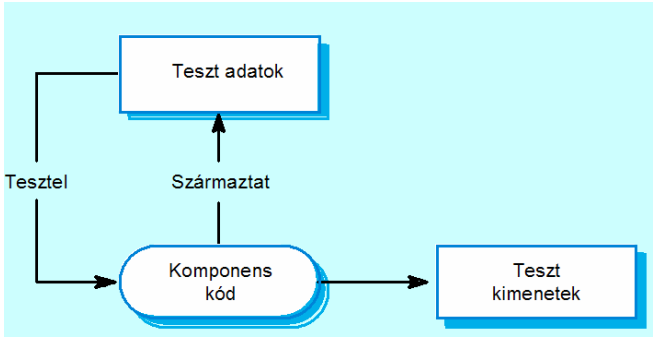
Sorozat	**Elem**
Egy érték	A sorozatban van
Egy érték	Nincs a sorozatban
Több, mint egy érték	A sorozat első eleme
Több, mint egy érték	A sorozat utolsó eleme
Több, mint egy érték	A sorozat középső eleme
Több, mint egy érték	Nincs a sorozatban

Bemenő sorozat (T)	**Keresett érték** (Key)	**Kimenet** (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

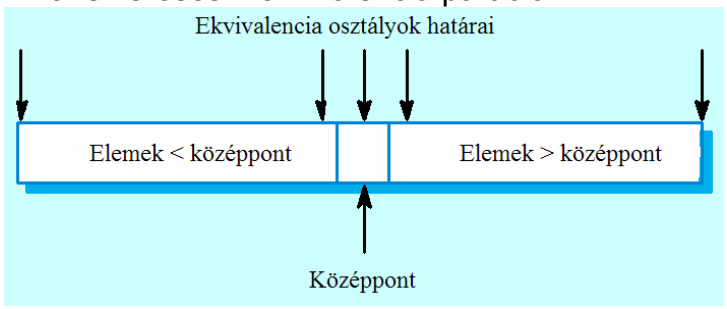
Strukturális tesztelés

- „Fehér doboz” tesztelésnek is nevezik.
- Teszt esetek a program struktúrája alapján. A program ismerete alapján újabb teszt esetek azonosítása.
- A cél valamennyi utasítás (de nem minden végrehajtási út kombináció) végrehajtása.

Strukturális tesztelés



Bináris keresés – ekvivalencia partíciók



Bináris keresés – ekvivalencia partíciók

- Előfeltételek kielégítve, keresett elem a tömbben.
- Előfeltételek kielégítve, keresett elem nincs a tömbben.
- Előfeltételek nincsenek kielégítve, keresett elem a tömbben.
- Előfeltételek nincsenek kielégítve, keresett elem nincs a tömbben.
- A bemeneti tömbnek egyetlen eleme van.
- A bemeneti tömbnek páros számú eleme van.
- A bemeneti tömbnek páratlan számú eleme van.

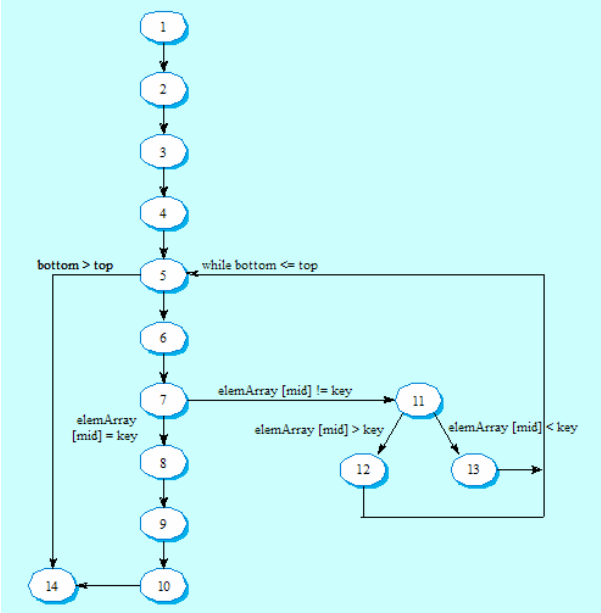
Bináris keresés – teszt esetek

Input array (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Végrehajtási út tesztelés

- A végrehajtási út tesztelés célja, hogy a tesztek minden végrehajtási utat legalább egyszer végrehajtanak.
- A kiinduló pont a program végrehajtási gráfja, ahol a csomópontok a program döntéseket, az élek pedig a vezérlés menetét jelképezik.
- A feltételes utasítások tehát csomópontok lesznek.

A bináris keresés végrehajtási gráfja



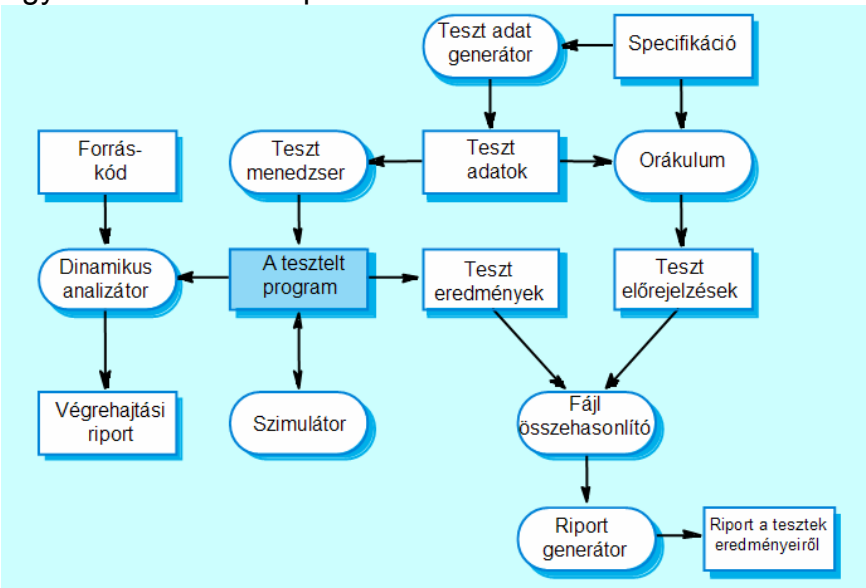
Független végrehajtási utak

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 5, 6, 7, 11, 13, 5, …
- Olyan teszteket kell generálni, hogy ezen útvonalak mindegyike végre legyen hajtva.
- Egy dinamikus program analizátor használatával ellenőrizhető, hogy minden út végre lett-e hajtva.

A tesztelés automatizálása

- A tesztelés nagyon drága fejlesztési fázis. A tesztelési munkapadok számos eszközt tartalmaznak, mellyel a tesztelési idő és költség redukálható.
- Egyes rendszerek (pl. JUnit) támogatják a tesztek automatikus végrehajtását.
- A legtöbb tesztelési munkapad nyitott rendszer, hiszen a tesztelési igények a szervezettől függenek.
- Ezeket időnként nehéz integrálni a zárt tervezési és analízis munkapadokkal.

Egy tesztelési munkapad



A tesztelési munkapadok adaptálása

- A felhasználói interfész szimulátorokhoz szkriptek, teszt-adat generátorokhoz minták fejleszthetők.
- A kimeneti értékek manuálisan előállíthatók összehasonlítás céljából.
- Speciális fájl-összehasonlítók fejleszthetők.

Összefoglalás

- A tesztelés felfedheti hibák jelenlétét a rendszerben, de nem tudja bizonyítani, hogy nem maradt több hiba.
- A komponensek fejlesztő felelősek a komponens tesztelésért, a rendszertesztelés egy független csoport feladata.
- Az integrációs tesztelés a rendszer növekményeinek tesztje, a végteszt pedig a megrendelőnek átadni kívánt rendszer tesztelésével foglalkozik.
- A hibatesztelés tervezéséhez mind a tapasztalat, mind ökölszabályok használhatók.
- Az interfész tesztelés feladata a kompozit komponensek interfészeiben levő hibák feltárása.
- A tesztek kidolgozásának egy módja az ekvivalencia partíciók kialakítása: a partíció olyan, hogy a benne lévő összes eset ugyanúgy viselkedik.
- A strukturális analízis a program analíziséből teszteket generál.
- Az automatikus tesztelés a tesztelési eljárást szoftver eszközökkel támogatva csökkenti a tesztelés költségeit.