MISKOLCI EGYETEM
GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

Szoftvertesztelés

Tesztelési- és teszt tervezési technikák

GEIAL31H-B

Tompa Tamás
egyetemi tanársegéd
Általános Informatikai Intézeti Tanszék

Miskolc, 2019

Tesztelési technikák

- **Statikus**
  - a szoftver forrás kódját vizsgálják **fordítási időben**
  - nem igényelnek futtatást
  - dokumentáció felülvizsgálat
  - Két típus
    - Felülvizsgálat
    - Statikus elemzés
- **Dinamikus**
  - a szoftver forrás kódját vizsgálják **futási időben**
  - tesztelendő rendszer futtatása
  - pl. komponens teszt
    - egységteszt

Statikus tesztelés

- **Felülvizsgálat**
  - kód, dokumentáció, vagy ezek együttes manuális átnézése
  - fehérdobozos teszt
    - gyanús részek keresése a kódban
  - Célja
    - szabványoktól / kódolási szabályoktól való eltérések keresése
    - követelményekkel kapcsolatos hibák feltárása
    - tervezési hibák beazonosítása
    - hibás interfész-specifikációk beazonosítása

Statikus tesztelés – felülvizsgálat típusok

- informális felülvizsgálat (csoporton belüli)
  - review
  - refactoring
  - pair programming

- átvizsgálás (házon belüli)
  - elkészült kisebb-nagyobb **modulok ismertetése a csapat többi tagjával** és a többi csapattal
  - célja, hogy a mások is átlássák az adott kódrészletet
    - adott programozó elvesztéséből fakadó kár csökkentése
    - megjegyzéseikkel segítik a kód minőségének javítását

Statikus tesztelés - felülvizsgálat típusok

- technikai felülvizsgálat (külsős szakérő bevonásával, rövid idejű)
  - a szoftver **teljesítményével való elégedetlenség**
  - **szűk keresztmetszetek keresése**
    - felhasználói visszajelzések alapján (profiler programok)
  - megoldás: nehézkes
    - ha lenne rá megoldás akkor azt alkalmazták volna…
    - külső szakértők bevonása
  - Célja tehát
    - a megtalálhatatlan hibák felderítése
    - a szoftver lassúságát okozó szűk keresztmetszetek megszüntetése
    - szabványok ellenőrzése

Statikus tesztelés - felülvizsgálat típusok

- inspekció (külsős szakérő bevonásával, hosszú idejű)
  - **hasonló mint a technikai felülvizsgálat, de:**
    - **a szakértőt adó cég részletesebb szerződést köt**, amely tartalmazza:
      - a megoldandó feladat leírását
      - azt a célfeltételt, ami a probléma megoldásával el kell érni
      - a célfeltételben használt metrikák leírását
      - az inspekciós jelentés formáját
  - technikai átnézésnél gyakran csak annyi elvárás van a szakértőktől, hogy legyen sokkal gyorsabb egy lekérdezés, az inspekció esetén **pontosan meg van határozva**, hogy milyen gyors legyen az

Statikus tesztelés - felülvizsgálat típusok

- inspekció (külsős szakérő bevonásával, hosszú idejű)
  - Inspektor
    - **nagy tekintélyű személy**
    - az általa javasolt változtatások általában fájóak
  - Jellemzők
    - a szoftvercég kezdeményezi
    - részletes szerződés szabályozza
    - inspekciós jelentés készítése,
    - célja teljesítmény fokozás a szakértő által kiválóan ismert technológia segítségével vagy elavult kód frissítése
    - opcionálisan PoC-ok (Proof of Concept) készítése
      - példaprogram csatolása

Statikus tesztelés – statikus elemzés

- **Statikus elemzés**
  - fehérdobozos teszt
  - forráskód szükséges
  - **null referencia hivatkozások kiszűrése**
  - például null referencia hivatkozás akkor lehetséges
    - ha egy „a = null;” értékadó utasítás és egy „a.akármi;” hivatkozás közt van olyan végrehajtási út, ahol az „a” referencia nem kap null-tól különböző értéket
    - lehetne dinamikus technikákkal is vizsgálni, de
      - annyi teszteset kell, ami minden lehetséges végrehajtási utat tesztel az „a = null;” és az „a.akármi;” között

Statikus tesztelés – statikus elemzés

- A következő hiba típusokat könnyebb statikus elemzéssel megtalálni, mint más technikákkal:
  - null referenciára hivatkozás
  - tömbök túl vagy alul indexelése
  - nullával való osztás
  - lezáratlan adat folyam (unclosed stream)
  - holtpontok (deadlock)
  - kiéheztetés (starvation)
- eszköz: pl. **FindBugs**
  - illesztés a fordítás folyamatába
  - plugin

Statikus tesztelés – statikus elemzés

- **FindBugs** (vagy SpotBugs)
  - telepíteni kell: Eclipse -> Help -> Install New Software…
    - vagy: https://marketplace.eclipse.org/content/spotbugs-eclipse-plugin
  - használat: projekt kiválasztása, majd a helyi menüben Find Bugs/Spot Bugs -> Find Bugs menü
  - megkeresi azokat a sorokat, amelyek valamilyen szabálynak nem felelnek meg
  - ha talál hibákat, akkor ezeket bal oldalon egy piros bogár ikonnal jelzi
  - hibákról részletes információ a FindBugs/SpotBugs perspektíva Bug Explorer ablakában
  - run automatically opció bekapcsolása
    - minden egyes mentésnél autómatikusan lefut

Statikus tesztelés – statikus elemzés

- **FindBugs**
  - Példa:

  - ```
    public int fact(int n) {
        return n*fact(n-1);
        }
    ```
    - „There is an apparent infinite recursive loop”
    - -> nincs bázisfeltétel, sosem áll le

  - ```
    Integer i = 1, j = 0;
    if(i == j) System. out.println(„egyezoek!");
    ```
    - „Suspicious comparison of Integer references”
    - tartalom egyenlőséget equals-el, ez itt így referencia egyezőség

Dinamikus tervezési technikák

- A **dinamikus tesztek tervezése** alapvetően az alábbi három lépésből áll
  - A tesztelés alanyának, céljának meghatározása (test condition)
  - Tesztesetek (test cases) specifikálása
  - Teszt folyamat (test procedure) specifikálása

Dinamikus tervezési technikák

- **A tesztelés alanyának, céljának meghatározása (test condition)**
  - rendszer egy **olyan jellemzője, amely ellenőrizhető egy vagy több teszt esettel**
  - Például
    - funkció
    - tranzakció
    - képesség (feature)
    - minőségi jellemző
    - strukturális elem

Dinamikus tervezési technikák

- **Tesztesetek (test cases) specifikálása**
  - végrehajtási prekondíciók (preconditions)
  - input értékek halmaza
  - elvárt eredmény
  - végrehajtási posztkondíciók (postconditions)
  - Folyamata:
    - a rendszert egy megadott kezdő állapotban kell hozni (prekondíciók)
    - megadott input értékek halmazával végre kell hajtatni a tesztelt elemet
    - a teszt futásának eredményét össze kell hasonlítani az elvárt eredménnyel
    - ellenőrizni kell, hogy a végrehajtás után a rendszer az elvárt állapotba (posztkondíciók) került-e

Dinamikus tervezési technikák

- **Tesztesetek (test cases) specifikálása példa**
  - program modul példa
    - amely a felhasználótól bekér néhány adatot
    - és megnyomja a „Számolj”gombot
    - a modul a megadott adatokat és adatbázisban tárolt egyéb értékeket felhasználva elvégez valamilyen számítást
    - majd az eredményeket adatbázisba menti
  - Ennek a modulnak egy tesztesete tartalmazza:
    - a felhasználói adatokat (input értékek halmaza)
    - a számítás helyes eredményét (elvárt eredmény)
    - prekondícióként azt, hogy a felhasználó megnyomta a „Számolj” gombot, és az adatbázis tartalmazza a számításhoz szükséges értékeket
    - posztkondícióként, hogy a számítás eredményei bekerültek az adatbázis megfelelő tábláiba

Dinamikus tervezési technikák

- **Teszt specifikáció**
  - **egy teszteset végrehajtásához szükséges tevékenységek sorozatának a leírása**
  - teszt forgatókönyv (manual test script)

- **Tesztkészlet**
  - **tesztesetek és hozzájuk tartozó teszt specifikációk halmaza**
  - csoportosítható egy teszt alanyra, vagy egy vizsgált hibára
  - archiválni kell, mert egy tesztkészletet a fejlesztés során többször is végre kell hajtani

Dinamikus tervezési technikák

- **Hibamodell**
  - **azon (feltételezett) szoftver hibák halmaza, amelyre a teszt tervezés irányul**
  - a tesztesethez kapcsolódó előbbi példához a hibamodell azt rögzítheti, hogy az alábbi hibák következhetnek be:
    - számítási hibák
    - adatbázis lekérdezési hibák (rossz adatokat használunk fel a számításhoz)
    - az adatbázisba módosításának hibák (a számítás eredménye rosszul kerül be az adatbázisba)

Dinamikus tervezési technikák

- **Teszt folyamat**
  - **egy rendszer teljes tesztelésének megtervezéséhez**
  - az alábbiakat foglalja magában:
    - a szükséges tesztelési célok meghatározása
    - minden tesztelési célhoz a szükséges tesz készlet definiálása
    - az egyes teszt készletekben foglalt tesztek ütemezésének és a végrehajtásuk dokumentálásának megtervezése

- **Teszt lefedettség**
  - **számszerű értékelése annak, hogy a tesztelési tevékenység mennyire alapos**, illetve hogy egy adott időpontban hol tart
    - „Már majdnem kész vagyok, főnök!” „Három hete ezen dolgozom, főnök!” - szubjektív mértékek

Dinamikus tervezési technikák

- **Teszt lefedettség (code coverage)**
  - tesztelési tevékenységet értékelésére az alábbi szempontok szerint:
    - lehetőséget ad a tesztelési tevékenység minőségének mérésére
    - megbecsüli mennyi erőforrást kell még a fejlesztési projekt hátralevő idejében tesztelési tevékenységre fordítani
  - **lefedettségi mérőszámok** tesztelési technikától függően eltérnek
  - milyen készültségi szinten áll a tesztelési tevékenység
  - milyen feltételek esetén tekinthetjük a tevékenységet késznek
  - Eclipse plugin: *EclEmma (JaCoCo)*

Teszt tervezési technikák

- **Specifikáció alapú technikák**
- **Modell alapú technika**
- **Struktúra alapú technika**
- **Gyakorlat alapú technika**

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - **tesztelés alapjaként a rendszer specifikációját tekintik**
  - teszteseteket a rendszer specifikációjából vezetik le
    - black-box: az egyes szoftver modulok belső szerkezetének ismerete nélkül, az egyes modulok által teljesítendő funkcionalitások alapján tervezik meg a teszteseteket
  - ha a specifikáció jól definiált és megfelelően strukturált
    - könnyen azonosítóak a tesztelés alanyai (test conditions), azokból pedig a tesztesetek
  - de általában a specifikáció nem azt rögzíti, hogyan kell a rendszernek megvalósítania az elvárt viselkedést (ez a tervezés feladata), csak magát a viselkedést definiálja

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Ekvivalencia particionálás (Equivalence partitioning)
  - Határérték analízis (Boundary value analysis)
  - Ok-hatás analízis (Cause-effect analysis)
  - Véletlenszerű adatok generálása
  - Használati eset (use case) tesztelés

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Ekvivalencia particionálás (Equivalence partitioning)
    - vannak olyan különböző input értékek, amelyekre a programnak ugyanúgy kell viselkednie
    - kimerítő tesztelés: minden lehetséges paraméterezésre teszteljük az outputot -> nem hatékony
    - ekvivalencia osztály: input értékek olyan halmaza, amelyre ugyanúgy kell viselkednie a programnak
    - egy ekvivalencia osztályhoz elég egy teszt esetet megtervezni és lefuttatni, mert az osztályhoz tartozó lehetséges tesztesetek
      - ugyanazt a hibát fedhetik fel
      - ha egy teszteset nem fed fel egy hibát, azt az osztályhoz tartozó más tesztesetek sem fog

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Ekvivalencia particionálás (Equivalence partitioning)
    - két elem kerüljön egy osztályba, ha a program a két értéken várhatóan nagyon hasonló módon fut le, majd elegendő csak az osztályok reprezentánsait tesztelni
    - ekvivalencia osztályok meghatározása heurisztikus
      - meg kell keresnünk az érvényes és az érvénytelen bemenetek osztályát is
      - szakterület és az adott feladat határozza meg, nincs általános módszer rá
    - ideális esetben az osztályok uniója kiadja az inputteret
    - csökkentheti a szükséges tesztesetek számát

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Határérték analízis (Boundary value analysis)
    - a határértékek kezelésénél könnyebben követnek el hibát a programozók, mint az „általános” eseteknél
    - célszerű tehát az **ekvivalencia osztályok határértékeit vizsgálni**
      - partíciók határain lévő értékek különlegesek lehetnek
    - Példa: program modul, amelynek feladata egy minta megkeresése egy sorozatban, tesztesetek:
      - 0 hosszúságú sorozat
      - 1 hosszúságú sorozat, a minta nincs benne / a minta benne van
      - >1 hosszúságú sorozat, a minta az első / utolsó helyen van
      - 2 hosszúságú sorozat (nincs benne / első /utolsó)
      - nagyon nagy elemszámú sorozat
      - Például String esetében: Üres, csak szóközök, null-referencia

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Határérték analízis (Boundary value analysis)

    Lépések:

    1. Feladat szakterületének megismerése
    2. Egy egységhez tartozó input/output értékek meghatározása
    3. Inputtér partícionálása, partíciók határelemeinek megvizsgálása
    4. Outputtér partícionálása, vizsgálata annak, hogy a program kimenetén megjelenhetnek-e a partíciók egyes elemei
    5. Általános tulajdonságok keresése a követelményekben, amelyek teljesülését ellenőrizni lehet
    6. Egységtesztek írása

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Ok-hatás analízis (Cause-effect analysis)
    - döntési tábla: oszlopai adják meg a definiálandó teszteseteket
    - specifikáció gyakran olyan formában írja le a rendszer által megvalósítandó üzleti folyamatokat, hogy az egyes tevékenységeknek milyen bemeneti feltételei vannak
      - Pl.:
      - egy input adat valamilyen értékére vonatkozó előírás
      - input adatok egy ekvivalencia osztálya
      - valamilyen felhasználói akció vagy egyéb esemény bekövetkezése
    - kimeneti feltétel (hatás): az okok egy kombinációjára a rendszernek milyen állapotot kell elérnie
    - bementi és kimenti feltételekhez logikai érték rendelhető
      - teljesül-e: igen/nem

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Ok-hatás analízis (Cause-effect analysis)
    - Példa:
      - egy áruház pontgyűjtő kártyát bocsát ki. Minden vásárló, akinek van ilyen kártyája, minden vásárlása során dönthet, hogy 5% kedvezményt kér a számla összegéből, vagy a kártyán lévő pontjait növeli meg. Az a vásárló, akinek nincs ilyen kártyája, szintén megkaphatja az 5% kedvezményt, ha 50.000 Ft felett vásárol
    - A bemeneti feltételek (okok):
      1. Van-e pont pontgyűjtő kártya?
      2. Kéri-e a kártyatulajdonos a kedvezményt?
      3. 50.000 Ft felett van-e a vásárlás összege?
    - A kimeneti feltételek (hatások):
      1. Nincs kedvezmény
      2. Kedvezmény jóváírása
      3. Pontok jóváírása

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Ok-hatás analízis (Cause-effect analysis)
    - A döntési tábla:

		T1	T2	T3	T4
	Okok:				
O1	Van-e pontgyűjtő kártya?	I	I	H	H
O2	Kéri-e a kártyatulajdonos a kedvezményt?	H	I	-	-
O3	50.000 Ft felett van-e a vásárlás összege?	-	-	H	I
	Hatások:				
H1	Nincs kedvezmény	I	H	I	H
H2	Kedvezmény jóváírása	H	I	H	I
H3	Pontok jóváírása	I	H	H	H

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Véletlenszerű adatok generálása
    - automatikusan, véletlenszerű bemeneti adatok generálása, majd ezekkel tesztelendő modul futtatása
    - adott tartományba eső (általában egyenletes eloszlású) számok generálása
    - Tulajdonságok:
      - kis erőforrás igény.
      - viszonylag könnyen automatizálható.
      - nagytömegű adattal tesztelhető a modul/rendszer.
      - "vak tyúk is talál szemet" elv alapján esetleg olyan hibára is fényt deríthet, amelyre a determinisztikus tesztek tervezése során nem gondoltunk
      - használható "monkey test"-ként, amellyel a próbálkozó felhasználó viselkedéséhez hasonló hatást lehet elérni
      - terhelési tesztre is alkalmas lehet

Teszt tervezési technikák

- **Specifikáció alapú technikák**
  - Használati eset (use case) tesztelés
    - use-case modell az alapja
    - teszteseteket leírása használati esetekkel
    - használati esetek és a tesztesetek összerendelése
      - hol tartunk a tesztelési folyamatban
      - teszt lefedettségi mátrixszal ábrázolható

	Használati esetek				
Test eset	1	2	3	4	5
1	■		■		
2		■			
3		■			
4		■			
5				■	■

Teszt tervezési technikák

- **Struktúra alapú technikák**
  - **alapja, hogy a programozási hibák gyakran a vezérlési szerkezeteket érintik**
  - a tesztelés célja tehát **a kód struktúrájának a felderítése** és helyességének ellenőrzése
    - a tesztesetek generálása a forráskód elemzése alapján történik
  - cél: **a vizsgált kód minden ágát végrehajtva vizsgálja annak működését**
    - kódbejárás alapja a kód matematikai modellje
      - **vezérlési folyamat gráf** (control-flow graph, CFG)

Teszt tervezési technikák

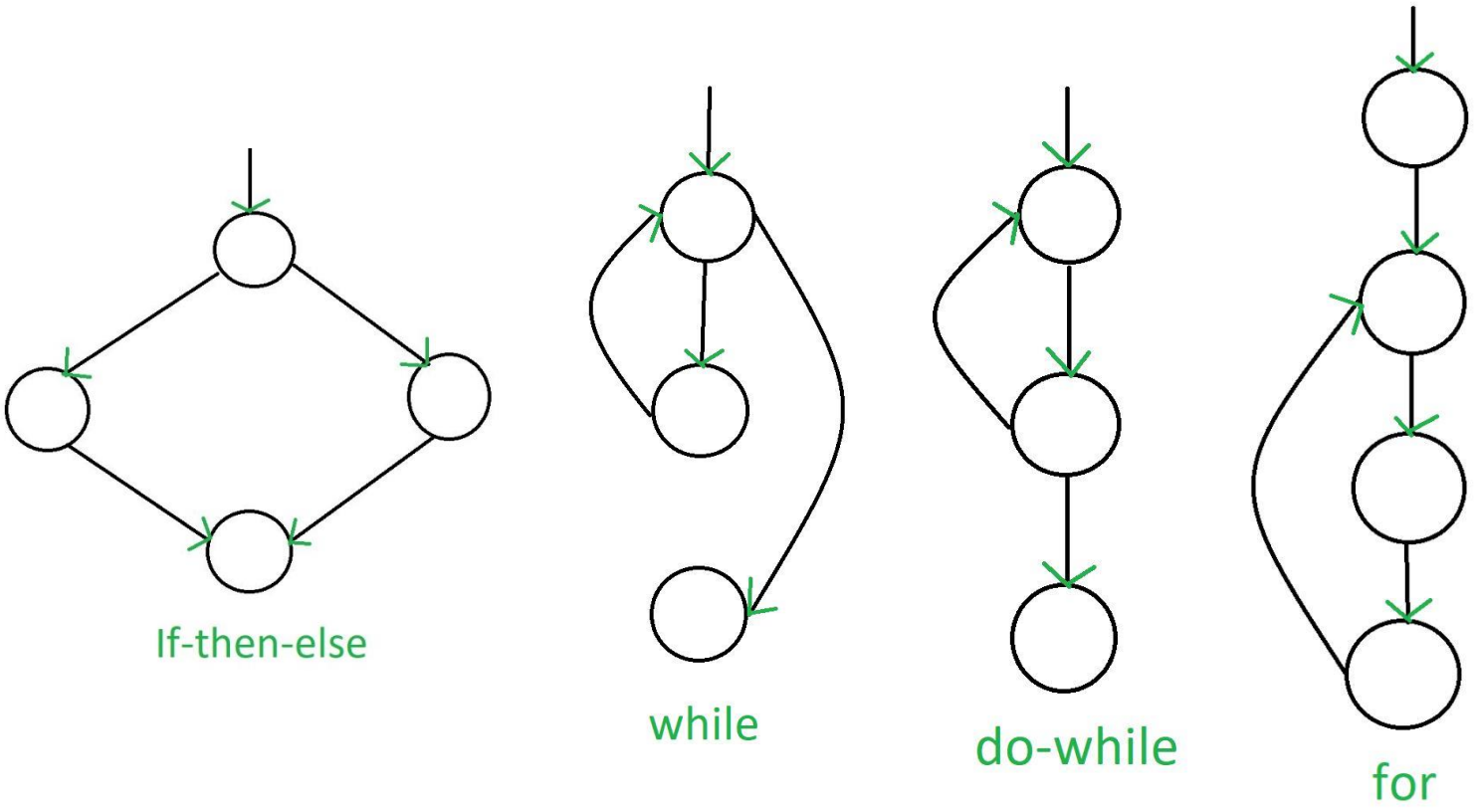
- **Struktúra alapú technikák**
- **vezérlési folyamat gráf (control-flow graph, CFG)**
  - irányított gráf
  - **csomópontok** a program utasításainak felelnek meg
  - az **irányított élek** pedig a végrehajtásuk sorrendjét jelölik ki
  - egy döntési utasításnak megfelelő csomópontból több él indul ki, a vezérlési ágak összekapcsolódási pontjában elhelyezkedő utasításhoz pedig több él fut be
  - a ciklust visszafelé irányuló él reprezentálja
  - a forráskódból automatikusan előállítható
  - **a teljes tesztelés valamennyi út bejárását jelenti**

Teszt tervezési technikák

- **Struktúra alapú technikák**
- **vezérlési folyamat gráf (control-flow graph, CFG)**
  - a feltételek nem mindig függetlenek egymástól, a bejárható utak száma általában kevesebb, mint az összes út
  - **ciklomatikus komplexitás (CK)**: a vezérlési gráfban megtalálható független utak maximális száma
    - két út független, ha mindkettőben létezik olyan pont vagy él, amelyik nem eleme a másik útnak
    - értéke arra jellemző, hogy a program vezérlési szempontból mennyire bonyolult
  - tesztelési **cél**: **a teszthalmaz fedje le a független utak egy maximális** (további független utakkal már nem bővíthető) **halmazát**

Teszt tervezési technikák

- **vezérlési folyamat gráf (control-flow graph, CFG)**



Teszt tervezési technikák

- **Struktúra alapú technikák**
  - **A strukturális tesztgenerálás lépései**
    - vezérlési gráf generálása
      - ez automatikusan végrehajtható a kód elemzésével
      - CK (ciklomatikus komplexitás) számítása
        létezik rá algoritmus, és a kód elemző eszközök képesek ezt az értéket meghatározni
        - Független utak maximális (CK db utat tartalmazó) halmazának generálása
    - bemenetek generálása a független utak bejárásához

Teszt tervezési technikák

- **Struktúra alapú technikák**
  - **A tesztelés alaposságának ellenőrzése kód lefedettségi mérőszámokkal**
    - számos mérőszám
      - elterjedtebbek: **utasítás lefedettség, ág lefedettség, út lefedettség**
    - **értéke általában 0 és 1 közé esik**
    - a teljes lefedettség sokszor csak irreálisan nagy teszteset halmazzal érhető el
    - a 100%-os lefedettség sem jelenti azt, hogy minden hiba megtalálsra került
    - **a különböző lefedettségi mérőszámok más és más szempontból értékelik** a tesztelés alaposságát célszerű többet is használni

Teszt tervezési technikák

- **Struktúra alapú technikák**
  - **Utasítás lefedettség**
    - **S(c) = s/S**
    - „s” a tesztelés során legalább egyszer végrehajtott, „S” pedig a program összes utasításainak a száma
    - 100% még nem biztosíték arra, hogy a teljes tesztelés minden hibát megtalál
    - Pl.:

```
int a=5 ;
x= …;
if (x>0) a = 10;
a = 20;
```

    - utasítás lefedettség: 100%: minden sorra rákerül a vezérlés
    - de: ha nem volt olyan teszt, amely során a feltétel igaz értéket vesz fel, nem derül ki a hiba

Teszt tervezési technikák

- **Struktúra alapú technikák**
  - **Ág lefedettség (döntés lefedettség)**
    - **D(c) = d/D**
    - „d” az elágazási utasításokban szereplő feltételek kimeneteinek tesztelés során bekövetkezett értékeinek száma, „D” pedig a program összes elágazás utasításaiban szereplő feltételeinek lehetséges száma
    - akkor teljes, ha a programban szereplő összes döntés minden lehetséges kimenete előfordult a tesztelés során
    - 100%-os lefedettség ugyan alaposabb tesztelést eredményez, de! -> példa

Teszt tervezési technikák

- **Struktúra alapú technikák**
  - **<u>Ág lefedettség (döntés lefedettség)</u>**
    - Pl.:

```
if (felt1 && (felt2 || fuggveny() ) )
   u1;
else
   u2;
```

    - teljes lefedettséghez két teszteset szükséges
      - egy feltételnek két lehetséges kimente van
    - ez a két teszteset lehet például:
      1. felt1 és felt2 igaz – ekkor az elágazás feltétele igaz,
      2. felt1 hamis, - ekkor az elágazás feltétele hamis.
    - ebben a két tesztesetben egyszer sem volt szükség a harmadik operandus kiértékelésére, tehát a függvény nem hívódik meg, ha abban van hiba, az felderítetlen marad

Teszt tervezési technikák

- **Struktúra alapú technikák**
  - **<u>Út lefedettség</u>**
    - **P(c) = p/P**
    - p a tesztelés során bejárt utak száma, P pedig a vezérlési gráf összes útjainak a száma
    - teljes út lefedettség teljes utasítás és ág lefedettséget biztosít
    - Nagyon szigorú mérőszám
      - 1. Az összes utak száma nagyon nagy lehet, ezért a tesztesetek generálása és lefuttatása erőforrás igényes
      - 2. A vezérlési gráfban lehetnek nem bejárható utak az egymást kizáró feltételek miatt, tehát a teljes lefedettség nem is mindig elérhető

Teszt tervezési technikák

- **Struktúra alapú technikák**
  - Tehát:
    - a struktúra alapú tesztelés bonyolult és erőforrás igényes feladat
    - végrehajtásához speciálisan erre a célra fejlesztett eszközök kellenek
      - manuális végrehajtása a bonyolult algoritmusok és a szükséges tesztesetek nagy száma miatt legfeljebb mintapéldákon lehetséges
    - de nem mellőzhetők ezek a tesztek a biztonság-kritikus rendszerek esetében

Tesztvezérelt fejlesztés

- **Test-Driven Development (TDD)**
  - a tesztelés és a kódfejlesztés folyamata együttesen, egymástól szét nem választható módon, párhuzamosan fejlődik
  - kód fejlesztése inkrementális módon történik, és mindig magával vonja az adott inkremens tesztjeinek a fejlesztését is
  - nem lehet a következő inkremens fejlesztésére lépni addig, amíg a korábbi kódok át nem mentek a teszteken
  - agilis szoftverfejlesztési módszertan

Tesztvezérelt fejlesztés

- **Test-Driven Development (TDD)**
  - 3 alapszabály
    - Tilos bármilyen éles kódot írni mindaddig, amíg nincs hozzá olyan egységteszt, amely elbukik
    - Tilos egyszerre annál több egységtesztet írni, mint ami ahhoz szükséges, hogy a kód elbukjon a teszt végrehajtásán. A fordítási hiba is bukásnak számít!
    - Nem szabad annál több éles kódot fejleszteni, mint amennyire ahhoz van szükség, hogy egy elbukó egységteszt átmenjen
    - -> Először mindig az egységteszt készül el, majd utána a kód, amit egységtesztelünk

Tesztvezérelt fejlesztés

- **Test-Driven Development (TDD)**
  - először mindig az egységteszt készül el, majd utána a kód, amit egységtesztelünk
  - amint az egységteszt kódja nem fordul le, vagy nem teljesül valamely állítása, az egységteszt fejlesztését be kell bejezni, és az éles kód írásával kell foglalkozni
  - de ilyen kódból csak annyit (azt a minimálisat) szabad fejleszteni, amely a tesztet lefordíthatóvá és sikeresen lefuttathatóvá teszi

  - Bringalámpa példa: célja jól világítson
    - először teszt írása, ami azt ellenőrzi, hogy jól világít-e -> teszt elbukik
    - a bringalámpa implementálása, de csak annyira, hogy átmenjen a teszten -> jól világítson
      - csak ezt tudja, semmi mást, nincs még 20 másik funkciója mert felesleges (nem kell, hogy lézercsíkot az útra vetítsen például)

Gyakorlat

Gyakorlat

TDD

Írjon olyan programot, amely teljesíti a következő tesztet!

```
1.  public class TeglalapTeszt {
2.      Teglalap TeglalapPOJO;
3.
4.      @Before
5.      public void init() {
6.          TeglalapPOJO = new Teglalap();
7.      }
8.
9.      @Test
10.     public void tesztSzamitTerulet() {
11.         assertEquals(1050, TeglalapPOJO.szamitTerulet(30, 35));
12.         assertEquals(1800, TeglalapPOJO.szamitTerulet(40, 45));
13.     }
14.     @Test(expected = IllegalArgumentException.class)
15.     public void tesztExceptionSzamitTerulet() {
16.         TeglalapPOJO.szamitTerulet(0, 35);
17.     }
18. }
```

Gyakorlat

- Egy program ami egy nevet (String) vár paraméterként, és egy üdvözlo szöveget ad vissza úgy, hogy a paraméter elé és összefűzi a "Hello, „ karakterláncot.
- Amennyiben a paraméter üres vagy null, a metódus IllegalArgumentException kivételt dob

- Egy lehetséges tesztelési terv?

Gyakorlat

- Tesztelési terv
  - Inputtér: sztringek
  - Outputtér: sztringek, kivétel
  - Particionálás: alfanumerikus, speciális karaktereket tartalmazó (ezen belül szóköz, sorvége, escape),...
  - Sztring határesetek: üres, szóközzel kezdődő/végződő, csak szóközökbol álló, nagyon hosszú string
- FindBugs/SpotBugs kipróbálása!

Gyakorlat

- **EmployeeDetails, EmpBusinessLogic, TestEmployeeDetails, TestRunner**
  - https://www.tutorialspoint.com/junit/junit_writing_tests.htm

Köszönöm a figyelmet!