

MISKOLCI EGYETEM
GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

Szoftvertesztelés
Teszt szintek

GEIAL31H-B

Tompa Tamás
egyetemi tanársegéd
Általános Informatikai Intézeti Tanszék

Miskolc, 2019



Teszt szintek

- Komponens teszt
- Integrációs teszt
- Rendszertereszt
- Átvételi teszt

Kialakulásuk okai

- A tesztelés korábban programozás utáni feladat
 - csakis az egyes funkciók tesztje
 - rendszer **működése közben hibák** felkutatása
 - komoly **többlemunka**
 - -> alacsonyabb szintű tesztelési technikák létrejötte
- Kiváltó okok:
 - rendszer legmagasabb szintű felhasználói felülete és az ott működő **folyamatok** összetett működést produkálnak, melynek **tesztelése** szintén **összetett feladat**

Kialakulásuk okai

- Sokszor egy-egy **újrafelhasznált egység** több funkcióban és részfunkcióban **okozhat helytelen működést**
 - egység módosítása után magasszintű tesztesetek elbuknak ránézésre érthetetlen okból
 - hosszas elemzés lehet szükséges ennek a kiderítésére
- A fejlesztési folyamat **késői fázisában derül ki a hibás működés**
 - minden egység vagy komponens fejlesztését meg kell várni ahhoz, hogy a felületen tesztelni lehessen

Kialakulásuk okai

- **A felhasználói felület tesztjei** gyakran **csak futás közben jelzik, hogy megváltozott az adott funkcionalitás** interfésze, struktúrája
 - ezzel szemben alacsonyabb szinten fordítási időben már kiderülhet az adott egység interfészének megváltozására.
 - egy fejlesztői IDE azonnal jelezni is tudja ezt a fajta változást
- **Alacsonyabb szintű tesztelési technikák** alkalmazása
 - -> nagyban csökkenthető a felhasználói folyamatok tesztjeinek elbukása

Komponens teszt

- a rendszer legkisebb, önállóan működő egységeinek tesztje
- **egy komponens tesztje önmagában**
 - általában a forráskód ismeretében (white box)
 - OO környezetben
 - osztályok
 - metódusok
 - eljárásorientált környezetben
 - eljárások
 - függvények

Komponens teszt

○ Fajtái

- **Egységteszt (unit test)**

- metódusokat teszteli
- adott paraméterekre ismert a metódus visszatérési értéke majd megvizsgálja, hogy a tényleges visszatérési érték megegyezik-e az elvárttal

- **Modul teszt**

- nem-funkcionális tulajdonságok tesztelése
 - sebességét
 - van-e memóriaszivárgás (memory leak)
 - van-e szűk keresztmetszet (bottleneck)

Egységteszt szabályok

- A következő minták, javaslatok helyesen alkalmazása kötelező!
- **Egy egységteszt pontosan egy programegységet tesztl önállóan**
 - mikor működik egy osztály önmagában?
 - szinte sosem...
 - nehéz elképzelni olyan architektúrát amely nem épít más rétegre vagy más eszközökre
 - Pl. rétegelt architektúra
 - -> függőségek lehetnek, lesznek
 - ezek viselkedését is kontrollálni kell

Egységteszt szabályok

- **Az egységtesztek nem lépik át saját moduljaik határát**
 - ha olyan programegységgel találkozunk, mely egy másik modul szolgáltatását használja, akkor valamilyen módon emulálni kell majd a másik modulban lévő szolgáltatás viselkedését

- **Az egységtesztek egymástól függetlenül működnek**
 - minden tesztnek megvan a maga elő- és végfeltétele
 - előfeltétel
 - futás előtt elérhetővé kell válnia
 - programozó állítja elő
 - végfeltétel
 - futás után meg kell felelnie

Egységteszt szabályok

- **Az egységtesztek egymástól függetlenül működnek**
 - egyik teszteset sem számíthat arra, hogy egy másik teszteset előállítja számára a kívánt előfeltételeket
- **Az egységtesztek a futtató környezetüktől függetlenül működnek**
 - ha valamilyen külső adatbázishoz vagy külső rendszerhez szeretnénk kapcsolódni, akkor állandó konfigurációra és/vagy emulációra lenne szükség a külső kapcsolatokat illetően

Egységteszt szabályok

- **Az egységteszteknek nincs mellékhatásuk**
 - futásuk után nem látunk változást a futtató környezetben, elsősorban a globális változókra kell figyelni ez esetben
- **Az egységteszteket úgy valósítjuk meg, hogy fordításkor futtathatóak legyenek**
 - azok a tesztek, melyek környezetfüggetlenek, és hatékonyan képesek lefutni, a fordító- és a teszt-keretrendszer képes futtatni bárhol, megállítva a fordítást, ha valamelyik teszt elbukik

Egységeszt szabályok

- -> az egységeszteknek szigorú **szabályoknak kell megfelelniük**
 - a szabályok betartásában segítenek a *stubbing* és a *mocking* technikák
 - a gyakorlatban sokszor inkább az definiálja majd a teszt fajtáját vagy szintjét, hogy e szabályok közül mit, milyen mértékben tartanak be

Egységteszt – AAA minta

- **A**rrange, **A**ct, **A**ssert
- Unit tesztek általános felépítése
- Arrange
 - értékek, adatok beállítása, inicializálása ami kell az adott tesztesethez.
- Act
 - tesztelt metódus meghívása
- Assert
 - viselkedés ellenőrzése

Egységteszt – AAA minta

```
1. //arrange
2. double r1 = 1.0;
3. double area1 = r1*r1*Math.PI;Circle
4. c1 = newCircle(newPoint(0.0,0.0),r1);
5.
6. //act
7. double resultArea = c1.Area();
8.
9. //assert
10. Assert.AreEqual(area1,resultArea,0.0001);
```

Integrációs teszt

- Progamegységeket, **modulokat, azok egymással és a környezettel történő együttműködését** teszteli
 - osztályok, **komponensek közötti interfészek, szolgáltatások és függőségeik** valós működését teszteli, ideértve az esetleges adatelérést is
- az **összeillesztés során keletkező hibákat keresi**
 - mivel a részeket más-más programozók, csapatok fejlesztették, ezért az **elégtelen kommunikációból súlyos hibák** keletkezhetnek
 - Pl. az egyik programozó valamit feltételez (pl. a metódus csak pozitív számokat kap a paraméterében), amiről a másik nem tud
 - elkerülés -> kontraktus alapú tervezéssel (design by contract)

Integrációs teszt

- érdemes minél hamarabb elvégezni
 - minél nagyobb az integráció mértéke, annál nehezebb meghatározni, hogy a fellelt hiba (általában egy futási hiba) honnan származik
 - ellenkező esetben, azaz amikor már minden komponens kész és csak akkor tesztelünk („nagy bummm teszt”, big bang teszt), akkor az rendkívül kockázatos

Integrációs teszt

- Tehát a szolgáltatások, logikák, modulok együttes működésének vizsgálata
 - előfordulhat olyan tesztek írására
 - amiben egy-egy programegység a valós működést hajtja végre
 - amiben a működést valamilyen módon emulálni kell
- Egységteszt
 - -> jól működő logika
- Integrációs teszt
 - logika perzisztens működése

Integrációs teszt - fázisok

- Az egységtesztetek követi
 - biztosítva van hogy a részegységek önmagukban már helyesen működnek

- Fázisok
 - technikai integrációs teszt (Integration Level Testing, ILT)
 - együttműködő egységek vizsgálata
 - részrendszer összeállítása már tesztelt elemekből
 - elsősorban verifikálás -> hibák megtalálására irányul

Integrációs teszt - fázisok

- rendszertereszt (System Level Testing , SLT)
 - a rendszer összes komponensének teljes körű (funkcionális, nem funkcionális) tesztelése
 - a rendszer kiadható-e a megrendelőnek
 - kettős cél:
 - Verifikáció: a rendszer olyan hibáinak megtalálása, amelyek az eddigi tesztelési tevékenységek során nem mutatkoztak meg
 - Validáció: főleg a nem funkcionális követelmények tesztelése segítségével meggyőződni arról, hogy a felhasználó céljainak megfelelő a rendszer működése
 - Több szempont
 - Szolgáltatás tesztelés
 - Mennyiségi tesztelés
 - Stressz tesztelés
 - Használhatósági tesztelés

Integrációs teszt - szempontok

○ Mennyiségi teszt

- nagy mennyiségű adattal történő tesztelés
- az adatmennyiség nem okoz-e hibás működést

○ Stressz-teszt

- a rendszer kitétele erős terhelésnek valamilyen szempontból
- válaszidők ellenőrzése

○ Használhatósági teszt

- egy meghatározott felhasználó által, egy meghatározott felhasználási körben használva
- meghatározott célok hatékony elérésére, mennyire kielégítő és mennyire vezet megelégedésre

Integrációs teszt - szempontok

- Biztonsági teszt
 - adatbiztonsággal és adatvédelemmel kapcsolatos hibák vizsgálata

- Konfigurációs teszt
 - különböző környezetek (hardver, operációs rendszer, egyéb szoftver installációk)

- Dokumentációs teszt
 - felhasználói és fejlesztési dokumentumokra
 - teljesség ellenőrzése
 - példák leképezése tesztesetekké majd azok végrehajtása

Integrációs teszt - fázisok

- elfogadtatási teszt (User Acceptance Testing, UAT)
 - a rendszer éles üzembe állítható-e
 - a felhasználó és minden haszonélvező (stakeholder) elégedettségének vizsgálata
 - a megrendelő telephelyén a végleges üzemeltetési körülmények között kell végrehajtani

Integrációs teszt - típusok

○ Típusai

- komponens – komponens integrációs teszt
 - a komponensek közötti kölcsönhatások tesztje a komponensteszt után
- rendszer – rendszer integrációs teszt
 - a rendszer és más rendszerek közötti kölcsönhatásokat tesztje a rendszerteszt után

Integrációs stratégiák

- Rendszerrendszerek összeépítésére és a tesztesetek megtervezésére és futtatására különböző stratégiák
 - „Bing-bang”
 - Inkrementációs
 - Top-down
 - Bottom-up

Integrációs stratégiák

○ „Bing-bang” integráció

- a rendszer **minden egység rendelkezésre áll** és ezekből **egyből a teljes rendszer építjük fel**
- Előnye
 - a teszteseteket könnyebben le lehet vezetni a követelmény analízisből
 - nagyon kevés segédkódot kell írni
- Hátránya
 - nagyon nehéz a hibák okát megtalálni
 - egy hibajelenséget több hiba együttese is okozhatja

Integrációs stratégiák

○ Inkrementációs integráció

- **a rendszer elemeit fokozatosan integrálja**
- minden egyes integrációs szinten teszteket hajt végre
- Lépési:
 - 1. **néhány elemet** (modult vagy részrendszert) **kombinál**
 - 2. olyan **teszteket** futtat, amelyek **csak az összeépített elemeket igénylik**
 - 3. ha minden **teszt sikeres**, **újabb elemeket** tesz hozzá a rendszerhez
 - 4. **további teszteseteket** tervez, amelyek az új elemek meglétét is igénylik
 - 5. minden eddigi **tesztet újra lefuttat**

Integrációs stratégiák

○ Top-down integráció

- 1. a hierarchia **legfelső szintjén álló elem** **tesztelésével kezd**
- 2. az egy szinttel lejjebb álló elemek viselkedését és interface-ét **szimuláló ideiglenes elemek** (stub) **szükségesek**
- 3. ha a **teszt sikeres**, az **ideiglenes elemeket a valódiakkal helyettesíti**, az általuk használtakat pedig újabb ideiglenes elemekkel szimulálja

Integrációs stratégiák

○ Bottom-up integráció

- 1. először a **legalsó szinten levő modulokat teszteli**, majd a hierarchiában **felfelé halad**
- 2. ehhez a **felső szinteket szimuláló tesztelési környezetet** (test driver) kell írni
- 3. ha a **teszt sikeres**, a **teszt driver-eket a valódi implementált elemekre cseréli**, és a következő szintet **helyettesíti test driver-ekkel**

Rendszerintegrációs teszt

- Vállalati környezet
 - -> több egymástól fizikailag is különálló alrendszert kell összekapcsolni
- E rendszerek tesztelése az integrációs **tesztek** egy speciális fajtájával, **az rendszerintegrációs teszttel**
- a **külső rendszerek** interfészeit **3rd party modul függőségként** kezeljük, és elrejtjük egy saját interfész mögé
 - a stubbing és a mocking technikák a külső rendszerek emulálásában segítenek

Integrációs teszt - függőségek emulálása

- függőségek **valódi példányának lecserélése** egy ugyanolyan interfésszel rendelkező, **általunk megvalósított** vagy konfigurált **másik példányra**
 - létező függőségek esetleges hibáit kiküszöbölése
 - hiányos vagy nem létező megvalósítások kiküszöbölése
- Egységtesztek esetében
 - minden függőséget kötelező emulálni
- Integrációs tesztek esetén
 - a teszteset és a függőségek készülsége határozza meg, hogy alkalmazunk-e valamilyen emulációs technikát vagy sem

Integrációs teszt - függőségek emulálása

- Csak saját típust emulálhatunk!
 - minden **3rd party eszközt**, API-t **saját interfész mögé kell rejteni**, majd függőségként annak egy megvalósítását használni a saját logika implementációjakor
- Pl.
 - Adatelérési réteg emulálása
 - **DAO (Data Acces Object) réteg bevezetése** -> adatbázis műveletek elfedése

Integrációs teszt - függőségek emulálása

○ Stubbing

- az adott függőség interfészét megvalósító saját típust készítünk, és az eredeti megvalósítást lecseréljük az általunk készített stub példányra
 - tesztelt programegység számára transzparens módon jelenik meg a stub viselkedés
- tetszőleges vizsgálatot, eredményt, viselkedést programozhatunk

Integrációs teszt - függőségek emulálása

○ Mocking

- stubbing-ból fejlődött
- stubbing technikával újra és újra megvalósított tesztelési részfeladatokat fogja össze és ad egy egységes keretet a tesztesetek megvalósításához
- speciális elvárásokat rögzít adott metódushívásokra, és definiálja, hogy bizonyos paraméterek esetén milyen eredményt adjuk vissza az adott metódus
 - a paraméter adatokra és a visszatérés eredményére helyeződik a hangsúly

Integrációs teszt - függőségek emulálása

○ Mocking

- keretrendszerek
 - Pl. **Mockito**
- a mocking keretrendszerek további eszközöket is adnak
 - viselkedéssel kapcsolatos elvárásokat is rögzíthetünk a keretrendszer eszközszerével
 - Pl. meghívódott-e a metódus bizonyos paraméterekkel vagy sem
 - hányszor hívódott meg (ellenőrzés, verification)
 - stb.

Rendszerteszt

- Egy szinttel fentebb
- **Teljes integrált rendszer tesztelése**
 - már **kész szoftverterméket teszteli**, hogy megfelel-e:
 - a követelmény specifikációnak
 - a funkcionális specifikációnak
 - a rendszertervnek
- Nincs emuláló technika
- A teljes rendszer olyan környezetben való tesztelése, ahol majd működni fog
- Gyakran feketedobozos teszt
 - külső cég végzi
 - Fejlesztők – Tesztelők között kapcsolattartás bug tracking rendszerrel

Rendszerteszt

○ Jellemzők

- Felhasználói felületen keresztül (GUI)
 - Kontrollok
 - Képernyő állapotok
 - Üzenetek, hibajelzés
 - Konzisztencia
 - Folyamatok és azok állapotai
- Input és output állományok
 - CSV
 - XML
 - PDF
 - Jelentések (reports)

Rendszerteszt

○ Tesztjelentések alapján

- a rendszer olyan állapotban van-e, hogy **átadható az ügyfélnek vagy sem**
- minden funkcionális igény tesztelésre került-e
- minden funkcionális igényt tárgyal és tartalmazza az ismert hibákat
- **döntések meghozatala** ezen dokumentum alapján

Rendszerteszt - automatizálás

- Tesztek automatikus futtatása
- Rendszer jellegétől függően
 - tesztek manuális futtatása
 - és bizonyos részekben automatizált futtatás
- Előnyök
 - a teszt futtatása gyors és hatékony
 - hosszútávon költség hatékony
 - érdekesebb, mint manuálisan kitölteni újra és újra a formokat
 - az eredmények azonnaliak és könnyen eljuttathatók az érdeklődőknek

Rendszerteszt - automatizálás

○ Hátrányok

- drága eszközök
- a tesztszkriptek fejlesztése miatt a tesztelés kezdetben kevésbé hatékony
- a tesztek karbantartása sok ráfordítással járhat
- a teszteszközök korlátokkal rendelkeznek

Rendszerteszt – speciális típusok

- Speciális rendszertesztek
 - Smoke-test
 - Sanity-teszt

Rendszerteszt – Smoke-test

- Hardver világból ered
- Bekapcsolás után lángralobban-e vagy sem
- Szoftvertesztelésben egy széles és sekély teszt gyűjteményt jelent
 - Széles: a rendszer minden funkcionalitásának érintése
 - Sekély: egyes funkcionalitások vizsgálata csak minimális ráfordítással
- Cél
 - pl. adatbázis séma helytelenségének felfedése
 - integrációs kapcsolatok, szolgáltatások működéséptelenségének felfedése
- Ha ez a teszt elbukik
 - -> alkalmazás architektúráisan alkalmatlan a működésre
 - -> az aktuális release visszautasítása

Rendszerteszt – Sanity-test

- Bizonyos módosítások után van-e egyáltalán értelme tovább tesztelni a rendszert
- Keskeny és mély teszt
 - a módosítások által érintett funkcionalitást vizsgálja részletekbe menően
- Általában manuálisan
 - egy új vagy módosult funkcionalitás ellenőrzése annyira, hogy belátható legyen, hogy a további tesztelésnek van értelme

Átvételi teszt

- Szintén az **egész rendszert teszteli**, de ezt már a **végfelhasználók végzik**
 - ügyfél vagy annak megbízottja végzi
 - fejlesztői csapattól független tesztelő
- Utolsó tesztelési szint
 - **elválasztja a rendszert az éles környezetben történő működéstől**
- Legismertebb fajtái
 - alfa teszt
 - béta teszt
 - felhasználói átvételi teszt
 - üzemeltetői átvételi teszt

Átvételi teszt

○ Alfa teszt

- a kész termék tesztje a fejlesztő cégnél, de nem a fejlesztő csapat által
- pl. össze-vissza kattintgatva ki lehet-e fektetni a programot (majom teszt)

○ Béta teszt

- a végfelhasználók egy szűk csoportja végzi
- pl. játékoknál: kiadás előtt néhány fanatikus játékosnak elküldik a játékot, akik rövid alatt sokat játszanak vele. Cserébe a felfedezett hibákat jelentése

Átvételi teszt

○ Felhasználói átvételi teszt

- majdnem az összes felhasználó, megkapja a szoftvert és az éles környezetben használatba veszi
- installálják és használják, de még nem a termelésben
- cél: a felhasználók meggyőződjenek, hogy a termék biztonságosan használható

○ Üzemeltetői átvételi teszt

- a rendszergazdák ellenőrzik
 - biztonsági funkciók, pl. a biztonsági mentés és a helyreállítás, helyesen működnek-e

Összefoglalás





Gyakorlat

Gyakorlat

- Hozzunk létre egy „User” nevű osztály
 - legyen egy „sum” nevű metódusa, amelynek paraméterei
 - user név (String)
 - user password (String)
 - két szám (int, int)
 - ha létezik az adott felhasználó az adott jelszóval akkor adja vissza a 2 szám összegét, egyébként null-t
- Hozzunk létre egy „DatabaseConnection” nevű interfészt
 - „checkUserPass” metódus
 - paraméterek: user neve (String) és jelszava (String)
 - visszatérés boolean
 - létezik-e a user vagy sem

Gyakorlat

- Ezek után a „User” osztályban hozzunk létre egy „DatabaseConnection” osztályváltozót
 - legyen olyan konstruktora a „User” osztálynak, amely beállítja a DatabaseConnection-t
 - nem implementáljuk az interfészt, majd valaki más fogja, nem tudni, hogy ki...
 - a „sum” metódusban mentsük el a DatabaseConnection visszatérését egy változóba
 - ha igazzal tér vissza csak akkor valósuljon meg a két szám összeadása

Gyakorlat

- Teszteljük a „User” osztály működését!
 - szükség van a „DatabaseConnection” interfész implementálásra
 - „User” osztály konstruktorában át kell adni
 - implementáljuk és adjon vissza true-t mindig
 - -> Fake object
- Teszteljük a „User” osztály működését mockito használatával!
 - Stub object
 - Mock object



Köszönöm a figyelmet!

thank you 😊