11. rész: Tesztelés. Verziókezelés

Bakay Árpád
NETvisor kft
(30) 385 1711
arpad.bakay@netvisor.hu

1. Hirdetmények

- 2. HF Beadási határidő: május 23
- Készül az Ursula V2.0 – legkésőbb holnap este

2. Entityk használata – tanulságok

- NE használjunk CHAR(x) Oracle adattípust
  - Az JPA nem tud rá rendesen search-elni
    - A JDBC igen!
  - Csak VARCHAR2-t
- A relációinkat persist előtt mindkét végén el kell rendezni!
  - A cache-elt objektumok relációit nem frissíti!
- Hogyan töröljünk egy teljes táblát?

DB törlés és cache ürítés - példa

```
@PersistenceUnit    private EntityManagerFactory emf; // factory-n keresztül
                      // alacsonyabb szintű, Oracle specifikus hozzáférés!!

public void ClearCache() {
   EntityManagerImpl emi = (EntityManagerImpl)emf.createEntityManager();
   emi.getServerSession().getIdentityMapAccessor().invalidateAll();
   emi.close();                                          // le kell zárni!
 }

public void ClearDB() {
            em. createNativeQuery("delete from ORVOS").executeUpdate();
            em.createNativeQuery("delete from OSZTALY").executeUpdate();
            em.createNativeQuery("delete from LABOR").executeUpdate();
            em.createNativeQuery("delete from ELLATO").executeUpdate();
            em.createNativeQuery("delete from USERACCOUNT").executeUpdate();
            ClearCache();
}
```

Session bean

- JAVA RMI-ben nincsenek out paraméterek
  - A megváltoztatott input parameter nem megy vissza a hívóhoz
  - Csak a return value
  - Több return value esetén:
    - Használjunk vmi Collectiont (pl. map)
    - Egyedileg definiált return osztályt
    - Vagy legyen több facade method

Futtatás, debuggolás
- Legfontosabb funkciók

- EJB oldalon: Debug - kliens osztályon: Run
  - Wiew/ Run manager : a futó / debuggolt processzek listája
- Breakpointok az EJB-ben is elhelyezhetők
  - Léptetés (F8), futtatás (F9), toggle (F5)
- Data inspector: View/Debugger/Data
  - ablak az adatok futás közbeni megfigyelésére
  - Közben az adatbázis is ellenőrizhető, ld. a Connections panelt
- „Hagyományos” eszközök, pl. System.out.println mindkét oldalon használhatók
  - Megjelenik az egyik messages ablakban
  - Console input viszont nincs!

3. Dokumentáció készítése - folyt

3/c. JavaDoc

- Kódba ágyazott dokumentációs kommenteket kell írni
  - Jdeveloper: a dekl. elemek előtti üres sorba szúrt „/**”-vel automatikusan
- Standard dokumentációs struktúra standard tag-ek alapján:

  @param [argument name] [argument description] - describes an argument of method or constructor.

  @return [description of return] - describes data returned by method (unnecessary for constructors and void methods).

  @throws [exception thrown] [exception description] - describes exception thrown by method.

  @author [author name] - identifies author(s) of a class or interface.

  @version [version] - version info of a class or interface.
- A „javadoc” pranccsal API dokumentáció generálható
  - JDeveloperben: Run/Javadoc
  - See also http://java.sun.com/j2se/javadoc/index.html

4. Kódolvasás, véleményezés

- „Peer review”: szakemberek átnézik és formálisan, tudatosan, szemérmeskedés nélkül értékelik egymás munkáját

- Bonyolult, embedded, R/T szoftver projekteknél különösen fontos
  - Miért? Mert itt lehetetlen mindent kitesztelni.
  - Eredete: NASA
  - Tárgya lehet több minden: kód, terv, dokumentáció
- Szerepek:
  - Szerző
  - Bírálók (előre átolvassák)
  - Moderátor
  - Jegyző
  - Hallgatók

…folyt

- Egy ülés keretében történik
  - hosszát kb. 90 percre kell tervezni
- Nem megoldást javasolnak, csak kérdeznek, ill. megjelölik a problémákat
  - Nem feltétlenül hiba, hanem bármi, ami nem tökéletes
  - A javítás viszont megint a szerző dolga
- Jegyzőkönyv készül az ülésről
  - Ebbe később az elvégzett javításokat is felveszik
- További előnye, hogy egységesíti a csoport programozási kultúráját
  - Strukturális/logikai ügyek mellett formai javítások is kérhetők

A formális code review csak nagy projektek esetén életszerű, de a kommunikáció, és egymás munkájának átnézése, ismerete mindig alapvető!

5. Tesztelés

- A tesztelés mélysége és mennyisége a minőségi szoftverfejlesztés legfontosabb ismérve!!!!
  - Pl: Agile/Xtreme programming: write tests first
- Két technika
  - Manuális tesztelés
  - Automatizált tesztelés
- Mindkét esetben előre definiált teszt esetek alapján
  - Ezen TestCase-ek gyűjteménye a TestSuite.

Manuális tesztelés

- Forgatókönyv
  - Minden testcase-t részletesen leír
- Tesztelési jegyzőkönyv
- Hibák visszajelzése a fejlesztőknek (lehetőleg automatikus feladatkezelő rendszerrel)

Manuális tesztelés forgatókönyve

- *Tesztelési forgatókönyv példa:*
  1. *Indítsa el a XXX programot*
  2. *Írja be a login adatokat: „rozi”, ill. „pw321’*
  3. *Nyomja meg a „Login” gombot*
  4. ***Ellenőrizze** a megjelenő képernyőt:*
     (Név: Róza Rozál, Szül.dátum: xxxx …. .)
  5. *Irja át a várost „Győr-ré”*
  6. *Lépjen ki.*
  7. *Újra nyomja meg a „Login” gombot*
  8. ***Ellenőrizze**, hogy a város „Győr”*

Tesztelési jegyzőkönyv (pl. Excel)

Alkalmazás: Verzió: Teszt ideje: Tesztelő:

Testcase	Ellenőrzés 1.	Ellenőrzés 2.	Ellenőrzés 3	Megjegyés
TC1114	OK	OK		
TC1115	OK	OK	OK	Sorok eltolódtak
TC1116	„Ő” betük kalaposak	OK	„Miskolc” jelent meg	

Atomatizált tesztelés

- Elsősorban regressziós tesztekhez
  - Tudja-e azt amit korábban tudott
    - Vagy: tudja-e már amit tudnia kéne
      - Ha előre megírtuk a TestCase-et – *extreme programming*
- Továbbá:
    - load testing: bír e 300 klienst?
    - performance testing: van-e olyan gyors, mint az előző verzió?
- Modul tesztek:

  Olyan kis kód-darabok, amelyek a tesztelt kódot hívják, és így alapvető ellenőrzéseket végeznek annak egyes osztályain, vagy osztályok kisebb csoportján.
    - Pl. egy Session EJB-n
    - Ajánlás:
      - Minden metódushoz legalább egy modul teszt-metódus
      - Minden osztályhoz legalább egy TestCase

JUnit

- Egy keretrendszer modul-tesztek írásához és futtatásához
  - Az teszteket csak meg kell írni, nem kell regisztrálni: „test*” nevű metódus egy önálló tesztnek számít, a framework gondoskodik a hívásukról.
  - A teszt metódusokban hívások a tesztelt kódra, ill. ellenőrző junit metódusok: *assertEquals(), assertNotNull(), assertFalse(), failNotSame()*
  - Az eredményeket könnyen ellenőrizhető formában megjeleníti
    - Pl. a végeredmény zöld vagy piros ikon vagy sáv
  - JDeveloperrel, Eclipse-szel tökéletesen integrált
    - Hogy még kényelmesebb legyen
- **Fő a maximális kényelem, különben senki sem fog teszteket írni!!!**

Előnyei, jellemzői

- Ingyenes!
- Az egyszer már megírt tesztekkel a tesztelés rendkívül fáradságmentes.
- A teszteket a fejlesztők írják
  - Nem objektív, viszont ők ismerik belülről a kódot
- A tesztek hierarchiába rendezhetők
  - testXxxx methods
  - YyyyTest class extends TestCase
  - ZzzzAllTests class – ú.n. test suite, tartalmaz tetszőleges számú testcase-t és testsuite-ot

JUnit tesztek felépítése, működése

- Teszt Case: pl. YyyyTest.java
  - extends *junit.framework.TestCase*
  - Jellemzően egy osztály tesztjei
  - run()-nál végighívja az összes *testXxxx()* metódusát
    - Ezek jellemzően a tesztelt modul hívásait és *assertPppp()*, *failPppp()* junit-hívásokat is tartalmaznak
  - Tesztek adatkörnyezete (az ú.n. fixture) a *setUp()* és *tearDown()* metódusokkal készíthető elő ill bontható le
- Test Suite: ZzzAllTests.java
  - Egy modul összes teszcase-je
  - run()-ra meghívja az összes gyermeke run()-ját
- A junit testrunner framework
  - Regisztrálja az összes teszt eredményét
  - A végén szépen megjeleníti azt

Tudta-e Ön?

- Hogy a Java-ban is van „assert” utasítás?

  http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html

Grafikus user interfész tesztelése

- A grafikus UI automatizált meghajtása
  - „Recording” vs. „Scripting” mód
  - Tájékozódás a UI ablakok, komponensek hierarchiájában.
  - Adatbevitel szimulálása az egyes widgeteken
  - Adatok kiolvasása és ellenőrzése más widgetekből
- Hagyományos és WEB-alkalmazásokhoz
  - HTTPUnit: *következő félévben*
- Problémák:
  - A vizuális benyomás ellenőrzése
  - Rajzolt grafikus területek ellenőrzése.
    - Fizikai pozícionálás az ablakokban…
    - Természetes vagy szintetizált képek grabbelése és ellenőrzése…

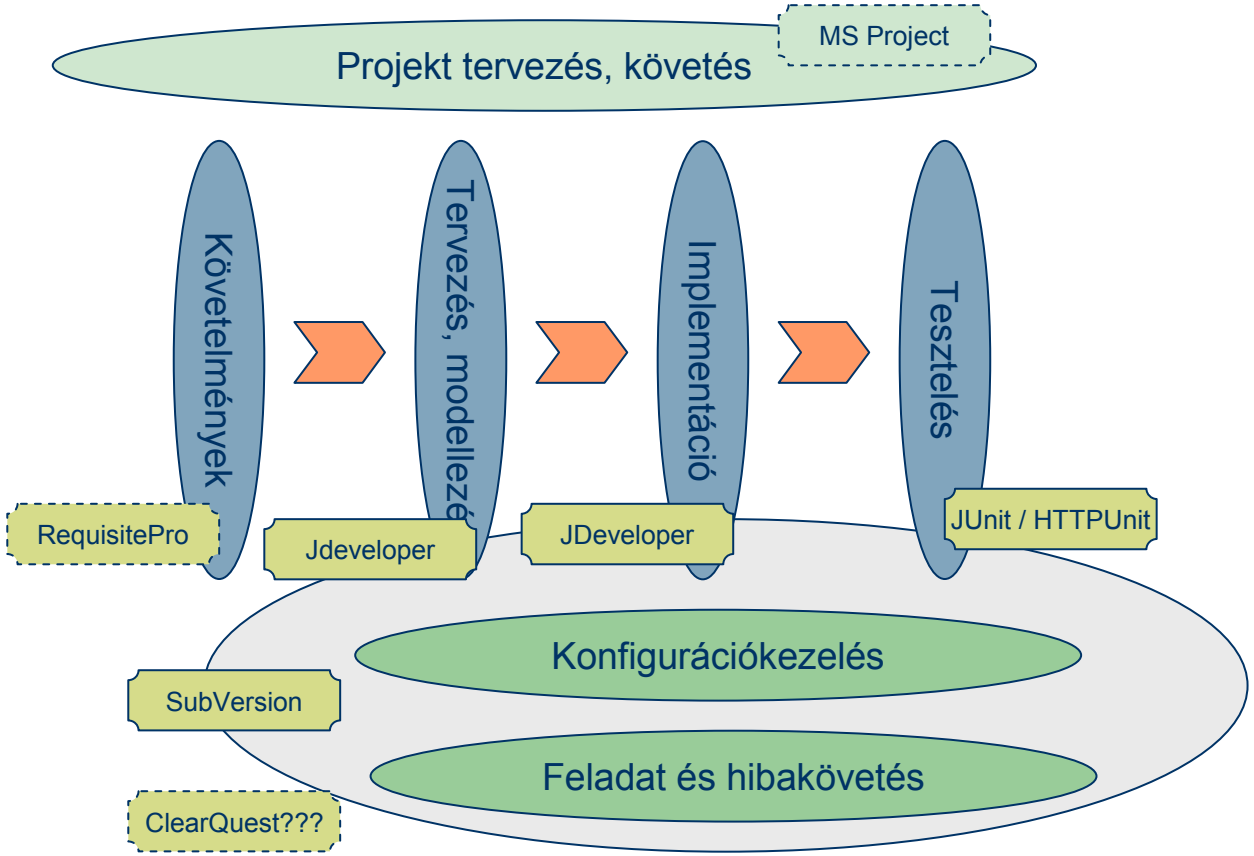
További minőségbiztosítási szempont: Tesztelhető és menedzselhető alkalmazás

- Beépíteni az alkamazásba a tesztelést segítő funkciókat (black-box -> gray-box -> white-box)
  - Assertion-ok
  - pl.: belső adatmodell dump-olása, vagy konzisztenciájának ellenőrzése
  - Cél interfészek erre a célra
    - RMI, SNMP, JMX, management konzol
  - Tesztelés szempontjából jól elkülönülő alkalmazás rétegek
- Öndiagnosztika hibák és problémák esetén
  - Az éles rendszerben is bőséges ellenőrzés és loggolás
  - Konfigurálható loggolási képességek
    - Java világban pl.: log4J

6. Verziókezelés és feladatkövetés

- Version control, code repository, software confugration management (SCM), software asset management (SAM)
- Bug/defect/change tracking/management,

Hol tartunk?


Projekt tervezés, követés
MS Project
Követelmények
Tervezés, modellezé
Implementáció
Tesztelés
RequisitePro
Jdeveloper
JDeveloper
JUnit / HTTPUnit
Konfigurációkezelés
SubVersion
Feladat és hibakövetés
ClearQuest???

1. Verziókövetés

Def: a szoftver megépítéséhez szükséges források tárolása és megosztása

- Biztonságosan
- Ellenőrzött hozzáféréssel
- A fejlesztési munka legújabb állapotának ill. korábbi mérföldköveinek megőrzésével.

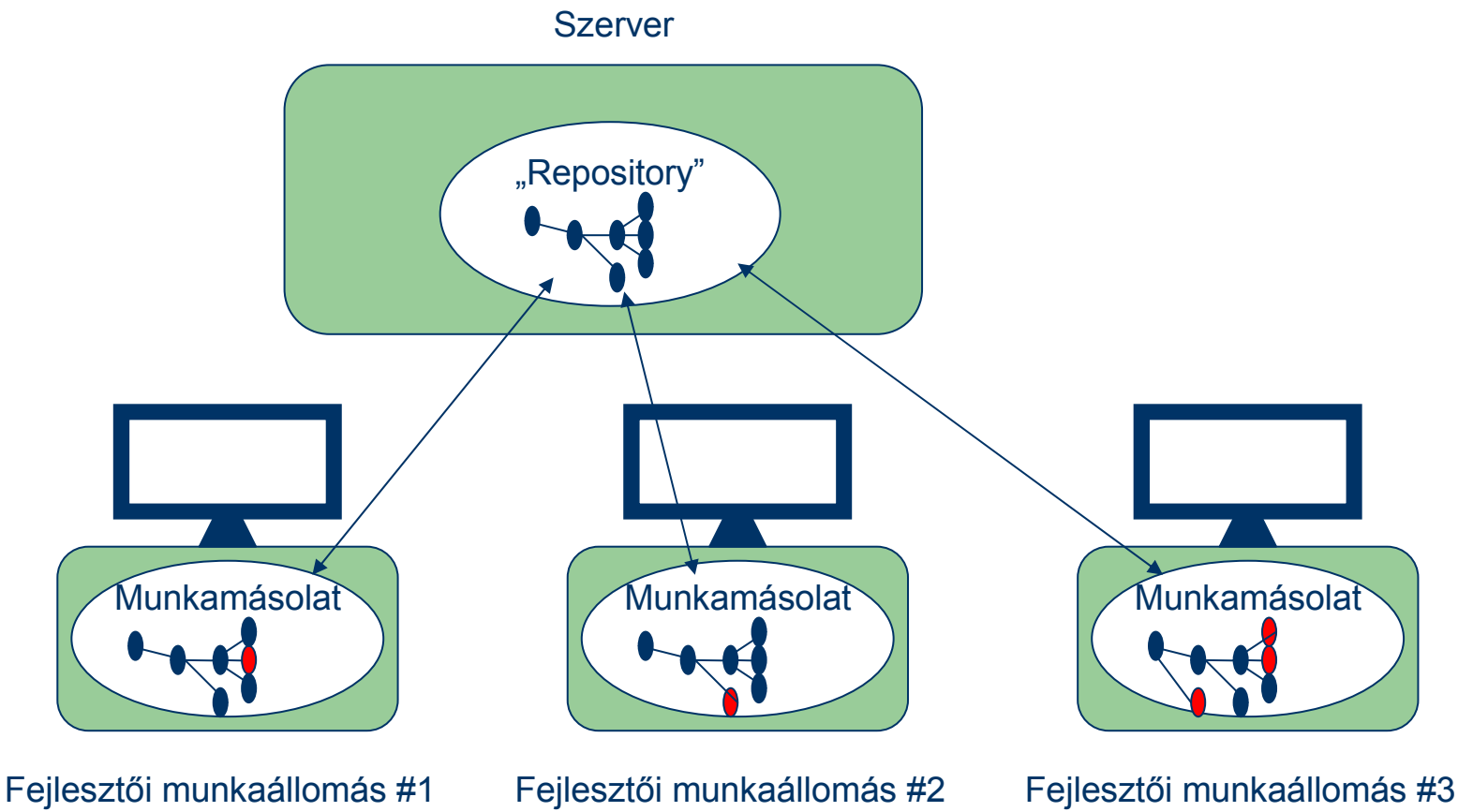
Biztonság

- Megbízható hardveren
  - Pl.: szerver tükrözött diszkkel
    - gyakorlatban: „workgroup server”
  - *diszk hiba ellen*
- Rendszeres mentés, archiválás
  - *szerver vagy környezeti hiba ellen*
- A fejlesztés teljes történetének rögzítése
  - Verziók azonosítása (pl. „revision number”)
  - *fejlesztői/kooperációs hiba ellen*
  - *új verziók és visszamenő javítások párhuzamos fejlesztéséhez*

Ellenőrzött hozzáférés

- Felhasználó azonosítás
  - Cél, hogy a változtatások névhez köthetők legyenek
  - Nem elsősorban a rosszindulatú támadók ellen!!
    - De: pl. a vétett hibák eltakarásának a szándéka előfordulhat
    - Esetleg: Ipari kémek ellen
- Konkurencia kezelés
  - File szinten (párhuzamos módosítás követése)
  - Alkalmazás logikai szinten
    - Konzisztens szoftver-állapotok megjelölése, megkülönböztetése a napi biztonsági mentésektől.

Verziókezelés architektúra



Verziókövetés alapműveletek

- Lokális munkamásolat készítése
  - „checkout” (CVS/SVN)
- Lokális másolat frissítése
  - „update” (CVS/SVN)
- Változások megtekintése
  - „log”, „diff”, „status” (CVS/SVN)
- Változások visszaküldése a repositoryba
  - „commit” (CVS/SVN)

A file szintű konkurrencia kezelése

- Optimistic concurrency
  - Mindenki írhat, mindent
  - Nem kell lock-olni
  - De: változások összefűzése szükséges lehet
    - „Merge”
- Strictly controlled concurrency
  - Megnyitás írásra művelet → kizárólagos
  - Check-In elvben mindig sikerül

A két megközelítés (és a terminológia) összehasonlítása

Munkalépés	Optimistic Példa: CVS, SVN	Pessimistic Példa: MS VSS
Lokális munkaverzió lekérése	„Checkout”, ”update”	„get latest version”
Lockolás írásra	Nincs	„checkout”
Párhuzamos változások egyesítése	„update” (with merge)	Nincs
Visszaküldés a szerverre	„commit”	„checkin”

Subversion

- Általános, ingyenes CMS eszköz
- Többféle hálózati protokoll felett:
  - Egygépes:
    - URL: file://
  - Kliens-szerver
    - http:// , https://
    - svn://
    - svn+ssh:// = svn ssh tunnel felett

SVN használata

- svn <cmd> <opts> <url>[@rev]

  svn checkout *svn://myrepos/myproj1/file.cpp*

  svn commit . –m „Bugs fixed”

  svn status myProg.java

  svn diff myProg.java

- Fejlesztői környezetbe integrált verziók

Köszönöm a figyelmet!

- arpad.bakay@netvisor.hu