Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Unit Testing

Lazar Maria-Magdalena (441A)

Nistorescu Andreea-Elena (441A)

Cuprins

1.Unit Testing (Lazar Maria-Magdalena)

|  |  |
| --- | --- |
|  | Beneficii |
|  | Gasirea problemelor mai rapid |
|  | Faciliteaza schimbarile |
|  | Documentatie |
| 1.1 | Design |
| 1.2 | Limitarile testelor unitare (Nistorescu Andreea-Elena) |
| 1.3 | Extreme programming |
| 2. Unit testing frameworks | |
| 2.1 | Visual Studio Unit Testing Framework |
|  | Clase de testare |
|  | Metode de testare |
|  | Asertii |
|  | Initializari si metode de cleanup |

3.Exemplu: Testare Unitara pentru Visual Studio 2010 (Lazar Maria-Magdalena si Nistorescu Andreea-Elena)

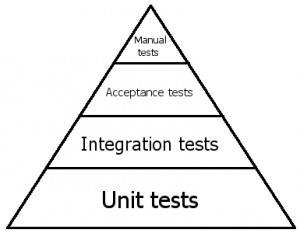
**1.Unit testing**

In programare unit testing reprezinta o metoda prin care fie unitati individuale de cod, fie seturi de unul sau mai multe programe sunt testate pentru a determina daca sunt gata pentru a fi utilizate .

Intuitiv putem sa percepem un unit ca pe cea mai mica parte testabila a unei aplicatii. In programarea procedurala un unit test poate reprezenta un modul intreg, dar este des intalnit ca o functie individuala sau o procedura. In programarea orientata pe obiecte un unit test reprezinta o intreaga interfata.

Testele unitare sunt bucati de cod relativ scurte ca dimensiune, create de programatori sau ocazional de white box testers in timpul procesului de development.

Ideal este ca fiecare test sa nu depinda de altul .Testele unitare sunt in mod normal scrise si rulate de software developers pentru a se asigura de faptul ca respectivul cod se ridica la asteptarile clientului si se comporta asa cum este specificat de catre client .



Ref:http://jonkruger.com/blog/2012/02/20/when-acceptance-tests-are-better-than-unit-tests/

**Beneficii**

Scopul testarilor unitare este de a izola fiecare parte din program si de a arata ca acele parti sunt corecte.Un test confera un contract scris , strict , care arata ca acea parte de cod trebuie sa fie corecta .

**Gasirea problemelor mai rapid**

Cu ajutorul testelor unitare se pot gasi problemele ce apar la nivelul codului mult mai rapid , inca din faza de development cycle .In test-driven development (TDD) , care este folosit frecvent si in Extreme programming cat si in Scrum , testele unitare sunt create inainte ca respectivul cod sa fie scris , iar atunci cand testele sunt trecute codul este considerat a fi complet .

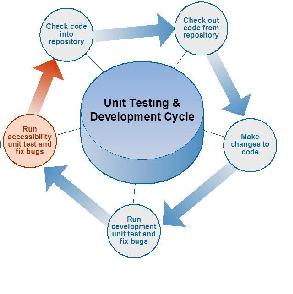
Aceleasi testari unitare sunt rulate pe aceeasi functie in timp ce aceasta este completata ( creste ca dimensiune ) sau schimbata . Daca rezultatul testarilor unitare nu este unul favorabil , atunci se considera a fi un bug in partea de cod schimbata sau chiar in testul aplicat .

Din moment ce testele alerteaza echipa de development de problemele care apar inaintea predarii codului catre tertari sau client in partea de inceput a procesului de development , se poate considera o metoda eficienta de inlaturarea a bug-urilor ce pot aparea.

**Facilitarea schimbarile**

Testarile unitare permit programatorului sa refaca codul la o data ulterioara, si sa se asigure ca modulul inca functioneaza corect (de exemplu, in testele de regresie). Procedura este de a scrie cazuri de testare pentru toate functiile si metodele, astfel incat ori de cate ori o schimbare cauzeaza o eroare, acesta poate fi identificata rapid. Testele unitare disponibile usureaza munca programatorlui de a verifica daca o bucata de cod se comporta in mod corespunzator.

In mediile de testare unitate continue acestea vor continua sa reflecte cu exactitate destinatia programului si a codul inaintea oricarei schimbari.



Ref: http://www.devmedia.com.br/introducao-ao-testes-unitarios-em-javascript/28236

**Documentatia**

Testele unitare ofera un fel de documentatie a sistemului cand ruleaza. Dezvoltatorii doresc sa invete cum este asigurata functionalitatea de catre o unitate de test si cum sa le foloseasca pentru a obtine o intelegere de baza a interfetei unitatii (API).

Cazurile de unit testing scot in evidenta caracteristicile care sunt critice pentru succesul programului. Aceste caracteristici pot indica utilizarea necorespunzatoare a unei unitati, precum si comportamentele negative care urmeaza a fi prinse de catre unitate.

Un caz de unit testing, in sine, documenteaza caracteristicile critice, desi multe medii de dezvoltare de software nu se bazeaza exclusiv pe cod pentru a documenta produsul in faza de dezvoltare.

Prin contrast, documentatia narativa este mult mai susceptibila de a deriva de la punerea in aplicare a programului si va deveni astfel depasita (de exemplu: modificari de proiect, practicile relaxate de mentinere a documentelor up-to-date).

**1.1 Design**

Atunci cand software-ul este dezvoltat folosind o abordare bazata pe test-driven , combinatia de scris a testului unitate pentru a specifica interfata plus activitatile de corectare efectuate dupa testul este trecerea, poate lua locul de proiectare formala.

Fiecare test de unitate poate fi vazut ca un element de design ce specifica clase, metode, si comportamentul observabil. Urmatorul exemplu Java va ajuta ilustrarea acestui punct de vedere.Aici este un set de cazuri de testare care specifica o serie de elemente de implementare. In primul rand, ca trebuie sa existe o interfata numita Adder si o clasa de punere in aplicare cu un constructor-argument la zero numit AdderImpl. Se merge pe principiul ca pentru a afirma ca interfata Adder trebui sa aiba o metoda numita add , cu doi parametri intregi, care returneaza un alt intreg. Acesta precizeaza, de asemenea, comportamentul aceastei metode pentru o gama restransa de valori intr-un numar de metode de testare.

In acest caz, testele de unitate actioneaza ca un document de proiectare specificand forma si comportamentul unei solutii dorite, dar nu si detaliile de implementare, care sunt lasate pentru programator.

**1.2 Limitarile testelor unitare**

Testarea nu va receptiona fiecare eroare din program , deoarece nu se poate evalua fiecare cale de executie , doar programele cele mai triviale . Acelasi lucru este valabil si pentru unitatea de testare . In plus , unitatea de testare prin definitie testeaza numai functionalitatea unitatilor de sine . Prin urmare , acesta nu va prinde erori de integrare sau erori la nivel de sistem ( mai largi , cum ar fi functiile indeplinite in mai multe unitati , sau zone de testare non- functionale , cum ar fi performanta ) . Unitatea de testare ar trebui sa fie facuta in legatura cu alte activitati de testare a software-ului , deoarece acestea pot arata doar prezenta sau absenta anumitor erori , ele nu pot dovedi o lipsa totala de erori . In scopul de a garanta un comportament corect pentru fiecare cale de executie si fiecare intrare este posibil si de a asigura lipsa de erori, sunt necesare alte tehnici , si anume punerea in aplicare a metodelor formale pentru a dovedi ca o componenta software nu are un comportament neasteptat .

De asemenea, este esential sa se puna in aplicare un proces durabil pentru a se asigura ca defectiunile sunt revizuite in fiecare zi si abordate imediat . In cazul in care un astfel de proces nu este pus in aplicare si se va inradacina in fluxul de lucru al echipei si astfel se va reduce eficacitatea lui.

Software-ul de sistem integrat de testare a unitatii prezinta o provocare unica : deoarece software-ul este dezvoltat pe o platforma diferita acesta nu poate rula cu usurinta un program de testare in mediul de implementare, asa cum este posibil cu programele de desktop .

**1.3 Extreme programming**

Extreme Programming foloseste crearea de teste unitare pentru dezvoltare condusa de teste . Dezvoltatorul scrie un test de unitate care nu expune nici o cerinta software sau un defect . Acest test va esua pentru ca nicio cerinta nu este implementat inca , sau pentru

ca expune in mod intentionat un defect in codul existent . Apoi , dezvoltatorul scrie un simplu cod pentru a face testul , impreuna cu alte teste .

Codul cel mai frecvent intr -un sistem este unit test-ul. Extreme Programming mandateaza un " test de tot ceea ce se poate rupe” , eventual, strategie , peste traditionale metode . Extreme Programming pur si simplu recunoaste ca testarea este rareori exhaustiva ( deoarece este de multe ori prea costisitoare si consumatoare de timp pentru a fi viabila din punct de vedere economic ) si ofera indrumari cu privire la modul de a se concentra in mod eficient resursele limitate .

.

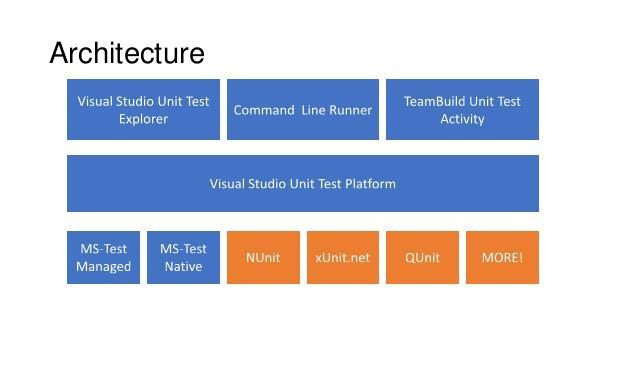
**2.Unit testing frameworks**

Cadrele de unitate de testare sunt cele mai multe ori produse de terti care nu sunt distribuite ca parte a suitei compilator . Ele ajuta la simplificarea procesului unitatii de testare , care a fost dezvoltat pentru o mare varietate de limbi . Exemple de cadre de testare includ solutii open source , cum ar fi condus - cod diferitele cadre de testare cunoscut colectiv ca xUnit , si solutiile proprietare / comerciale , cum ar fi TBrun , JustMock , Isolator.NET , izolator + + , Parasoft Dezvoltare Testare ( Jtest , Parasoft C / C + + testare , dotTEST ) , Testwell CTA + + si VectorCAST / C + + .

In unele cadre ale unitatii de testare mai multe caracteristici avansate lipsesc sau trebuie sa fie scrise de programator .

**2.1 Visual Studio Unit Testing Framework**

Visual Studio e un cadru de testare a unitatii ce descrie suita Microsoft de instrumente, este integrat in unele versiuni ale Visual Studio 2005 si mai tarziu aparute.. Teste unitare create cu cadrul de unitate de testare pot fi executate in Visual Studio sau, folosind MSTest.exe, de la o linie de comanda.



Ref: http://pt.slideshare.net/abhimanyusinghal/unit-testing-with-visual-studio-2012-18774256

***Clase de testare***

Clasele de testare sunt declarate in asa fel incat sa se utilizeze TestClass attibutes . Aceste atribute identifica clasele care contin metode de test . Cea mai buna metoda este cea in care o clasa de test contine un singur test unitate .

***Metode de testare***

Metodele de testare sunt declarate astfel incat o clasa sa contina TestMethod arrtibute. Acest atribut este utilizat sa identifice metodele care contin teste unitate .

Cea mai buna metoda este ca metodele de test sa contina un singur test de unitate.

***Asertii***

O asertie este o bucata de cod care ruleaza pentru a testa o conditie sau un comportament si compara rezultatul cu unul asteptat de catre programator. In Visual studio se gasesc sub numele de Assertion class.

***Initializari si metode de cleanup***

Sunt utilizate pentru a pregati testele unitare inainte de a rula si “cleaning up” dupa ce s-au executat.Metodele de initializare sunt declarate cu ajutorul TestInitialize attribute , iar cele de cleanup cu ajutorul a Testcleanup attribute.

**3. Exemplu: Testare Unitara pentru Visual Studio 2010**

Pentru inceput, vom crea un nou proiect. Acesta fiind un proiect ce va contine codul ce dorim a fi testat. Acum e momentul sa introducem un proiect de test. O sa creem un nou proiect de tip Unit Test Library. In unit test o sa observam ca avem urmatoarele tipuri de etichete si metode:

* Eticheta **[TestClass]** este obligatorie inaintea clasei din proiectul de testare.
* Eticheta **[TestMethod]** este de asemenea obligatorie inaintea metodelor ce reprezinta testarea propriu-zisa. In plus, metodele de testare trebuie sa fie de tipul *void* sisa nu aiba parametri.
* Se creaza un context in care cunoastem rezultatul si il comparam cu raspunsul dat doar de partea care ne intereseaza din proiect, si verificam folosind utilitarul

**Assert.AreEqual();**

Pentru a continua, va trebui sa adaugam o referinta catre namespace-ul using Microsoft.VisualStudio.TestTools.UnitTesting.

In continuare ne vom folosit de fereastra Test Explorer din cadrul procesului de Build, de un avem posibilitatea de a alege testele pe care dorim sa le executam. O sa observam de asemenea langa teste si daca aceastea au fost trecute sau nu in cazul in care am dorit sa le efectuam. Microsoft Visual Studio Unit Testing Framework ne pune la dispozitie o gama variata de unelte pentru a specializa pe cat posibil testarea unitara.

Daca functia pe care dorim sa o testam e scrisa sa arunce exceptii, putem verifica si aceasta actiune. Inaintea declaratiei functiei de testare, dupa eticheta **[TestMethod]**, introducem eticheta care indica modulului de testare tipul exceptiei asteptate. La fel, in interiorul metodei cream un context care stim sigur ca ar trebui sa arunce o exceptie si apelam metoda, numai ca de aceasta data nu mai folosim utilitarul **Assert**, de acest lucru se ocupa automat eticheta **ExpectedException.**

Bibliografie:

* http://en.wikipedia.org/wiki/Unit\_testing
* https://msdn.microsoft.com/en-us/library/aa292197%28v=vs.71%29.aspx
* http://en.wikipedia.org/wiki/List\_of\_unit\_testing\_frameworks
* https://msdn.microsoft.com/en-us/library/hh694602.aspx
* https://msdn.microsoft.com/en-us/library/ms243147%28v=vs.80%29
* http://www.onjava.com/pub/a/onjava/2003/04/02/javaxpckbk.html
* http://www.extremeprogramming.org/rules/unittests.html