Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Gorjan

# Primerjava ogrodij za testiranje enot v programskem jeziku C#

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: dr. Igor Rožanc

Ljubljana 2015

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Testna ogrodja pomembno vplivajo na učinkovitost izvedbe testiranja programske opreme. V diplomski nalogi predstavite testiranje enot v jeziku C# in tri izbrana ogrodja: NUnit, xUnit in MSTest. Poleg predstavitve uporabe ogrodij opravite objektivno primerjavo teh na podlagi večjega števila kriterijev ter predlagajte najprimernejše ogrodje za razvijalca programske opreme v okolju .NET.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Klemen Gorjan, z vpisno številko **63100114**, sem avtor diplomskega dela z naslovom:

*Primerjava ogrodij za testiranje enot v programskem jeziku* `C#`

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Igorja Rožanca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 22. januarja 2015 Podpis avtorja:

*Zahvalil bi se mentorju viš. pred. dr. Igorju Rožancu za nasvete in popravljanje napak pri izdelavi diplomskega dela. Posebej se zahvaljujem tudi staršem za podporo pri študiju.*

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **HTML** | Hyper Text Markup Language | Označevalni jezik za izdelavo spletnih strani |
| **XSLT** | Extensible Stylesheet Language Transformations | Razširljiv jezik za pretvorbo XML dokumentov |
| **XML** | Extensible Markup Language | Razširljiv označevalni jezik za izmenjavo strukturiranih podatkov |

# Povzetek

Diplomsko delo se posveča prvemu nivoju v procesu testiranja programske opreme - testiranju enot. Primerjavo bomo izvedli med tremi ogrodji za testiranje enot, ki jih lahko uporabimo v programskem jeziku `C#` - ogrodje NUnit, xUnit in MSTest. Najprej se bomo bolje spoznali z osnovnimi koncepti in predstavili različne nivoje testiranja. Sledilo bo podpoglavje o testiranju enot, kjer bomo predstavili pomembne izraze, ki so potrebni za razumevanje diplomske naloge in kakšne prednosti oz. slabosti nam testiranje enot sploh prinaša.
Vsa tri ogrodja smo temeljito testirali na podlagi sledečih kriterijev: obstoj dokumentacije, popularnost, integracija z razvojnim okoljem Visual Studio, izdelava poročil, možnosti konfiguracije testov, trditvene metode, način izvajanja testiranja, razvoj ogrodja in uporabnost v praksi. Kot najpomembnejši kriterij smo izbrali način izvajanja testiranja, saj lahko življenjski cikel testnega razreda vpliva na rezultate testnih metod, zato je izjemno pomembno da poznamo razliko med ogrodji. Glede na podane kriterije se je kot daleč najboljše ogrodje izkazal xUnit, ogrodji NUnit in MSTest pa sta si bili zelo blizu.

**Ključne besede:** testiranje programske opreme, testiranje enot, ogrodje, C#, NUnit, xUnit, MSTest.

# Abstract

The main focus of this thesis is on the first level of software testing - unit testing. The comparison will be made between three frameworks for unit testing in `C#` programming language - framework NUnit, xUnit and MSTest.
At first, we will get acquainted with basic concepts and different levels of testing. Then we will focus on unit testing, where we will present some important terms, which are important for understanding the thesis. At the end of the chapter, the benefits and disadvantages will be explained.
All three frameworks were thoroughly tested based on these criteria: the existence of documentation, popularity, integration with Visual Studio development environment, report creation, possibility of test configuration, assert methods, method of testing implementation, development of the framework and usefulness in practice. Method of testing implementation was chosen as the most important criteria, because life cycle of the test class can affect the results of the test methods, so it is very important that we know the difference between different frameworks. Based on given criteria, framework xUnit was proved as the best, whereas NUnit and MSTest were very close.

**Keywords:** software testing, unit testing, framework, C#, NUnit, xUnit, MSTest.

# Poglavje 1

# Uvod

Ljudje vedno več časa preživimo pred računalniškimi zasloni, a se malokrat zavedamo koliko truda je bilo vloženega v razvoj programov ki jih uporabljamo. Od njih zahtevamo vedno več, kar posledično pomeni kompleksnejši razvoj programske opreme. Breme pade neposredno na razvijalce, katerih dolžnost je razviti kar se da kvaliteten in zanesljiv produkt. Vse to pa ni mogoče brez aktivnosti, ki se imenuje testiranje.

Testiranje enot predstavlja le en del aktivnosti, ki pa je obvezna podlaga za testiranje na ostalih nivojih. Posvečamo se najmanjšim delom programske kode, za katere želimo da brezhibno opravljajo svojo funkcijo. Izvajanje testov in pisanje le-teh, ki so dolžnost razvijalca, praviloma poteka vzporedno s samim razvojem programske opreme. Testiranje enot nam omogoča posebna programska oprema, ki ji pravimo ogrodje za testiranje enot.

Diplomska naloga se tako v večji meri posveča trem različnim ogrodjem. Izmed vseh, smo si po našem mnenju izbrali tri trenutno najbolj zanimiva in popularna, ki nam omogočajo testiranje enot v programskem jeziku `C#`. To so ogrodja NUnit, xUnit in MSTest. Ker je izbira lahko zelo zapletena, smo najprej določili kriterije, po katerih bomo čimbolj objektivno primerjali vsa tri ogrodja ter tako predstavili razlike med njimi. Kriterije bomo tudi različno utežili, kar predstavlja njihovo pomembnost. Glede na končne rezultate bo sledila proglasitev najbolj primernega ogrodja za testiranje enot v programskem jeziku `C#`.

# Poglavje 2

# Testiranje programske opreme

Tehnologija je živa, iz dneva v dan spreminjajoča se stvar. Programska oprema postaja zahtevnejša, s samo zahtevnostjo pa se veča tudi možnost napak. Ker naročnik od končnega izdelka pričakuje kakovost in zanesljivost, je pri razvoju programske opreme potrebna dodatna aktivnost, ki ji pravimo testiranje (*ang. testing*). Gre za proces, ki se izvaja vzporedno s samim razvojem programske opreme in zagotavlja, da program deluje v skladu s pričakovanji in zahtevami [1].

Zaradi vse večjega zanimanja o testiranju programske opreme, se na internetu pojavljajo zanimive teme med pripadniki in nasprotniki testiranja. Tako smo ob prebiranju odgovorov naleteli na nekaj zanimivih citatov, ki so se nam zdeli vredni omembe:

“Kvaliteta se ne zgodi; vedno je rezultat pametnega načrtovanja.“ - John Ruskin
“Kvaliteta je zastonj, ampak samo za tiste, ki so jo pripravljeni drago plačati.“ - T. DeMarco and T. Lister
“Če sam ne želiš testirati programske opreme, je zelo verjetno tudi uporabniki ne bodo želeli.“ - Anonimno
“Preizkuševalci programske opreme uspejo, kjer drugim spodleti. “ - Anonimno
“Edine gotovosti v življenju so smrt, davki in napake v kodi.“ – Anonimno [4]

## 2.1 Osnovna terminologija

Za začetek se najprej seznanimo z nekaterimi osnovnimi, a pomembnimi pojmi:

**Validacija** (*ang. validation*): Zagotovitev, da produkt oz. sistem dosega standarde in zahteve, ki so postavljeni s strani naročnika. Gre za proces, ki ponavadi vključuje stranko in se izvaja na koncu razvoja programske opreme [15].

**Verifikacija** (*ang. verification*): Ocena, ali je produkt oz. sistem skladen s predpisi, zahtevami, specifikacijo. Za razliko od validacije, je verifikacija interni proces [15].

**Napaka** (*ang. fault*): Hiba, pomanjkljivost v programu ali sistemu, ki povzroči nepravilen ali nepričakovan rezultat. Večina napak izvira iz napačno razumljenih zahtev naročnika.

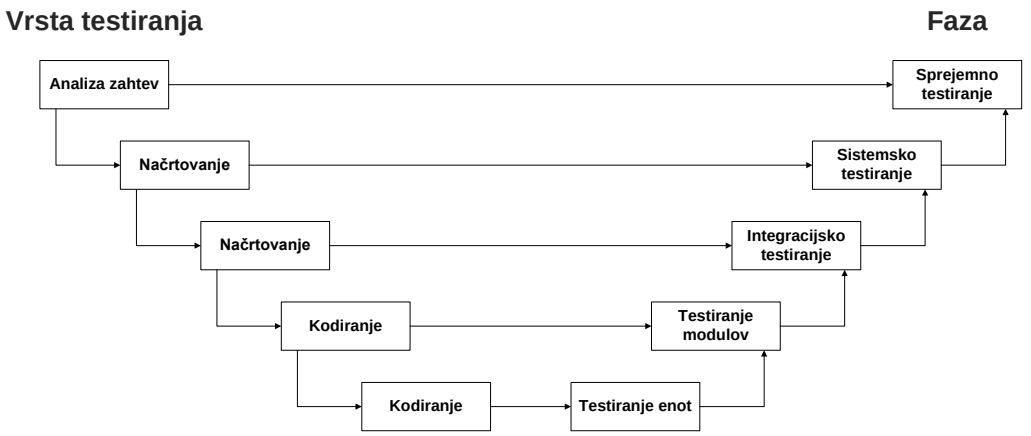
**Okvara** (*ang. error*): Posledica, ki lahko nastane zaradi napake v programu. Vsaka napaka ne povzroči okvare.

**Odpoved** (*ang. failure*): Nezmožnost nadaljnega izvajanja programa zaradi napake v izvorni kodi. Vsaka okvara ne povzroči odpovedi.

**Testiranje** (*ang. testing*): Metoda izvajanja programske opreme z namenom zagotovitve pravilnega delovanja.

**Testna odpoved** (*ang. test failure*): Izvajanje programa s testnimi podatki, ki se konča z odpovedjo.

**Razhroščevanje** (*ang. debugging*): Proces lociranja in odprave napak, ki smo jih odkrili s testiranjem. Razhroščevanje se začne z napako, kateri sledi osamitev napake in nato odprava le te. V ta namen se uporabljajo posebna orodja, ki jim pravimo razhroščevalniki (*ang. debuggers*) [10].




Slika 2.1: Ravni testiranja glede na aktivnost razvoja

## 2.2 Nivoji testiranja in izvajalci

Tako razvoj programske opreme kot testiranje le te je večfazni proces. Zatorej je teste smiselno razdeliti na več nivojev, kjer vsak nivo razločno predstavlja določeno aktivnost v fazi razvoja programske opreme (slika 2.1). Ravni testiranja si od vrha proti dnu sledijo takole:

- sprejemno testiranje (*ang. Acceptance testing)*,
- sistemsko testiranje (*ang. System testing)*,
- integracijsko testiranje (*ang. Integration testing)*,
- testiranje modulov (*ang. Module testing)* in
- testiranje enot (*ang. Unit testing)*.

Glavni cilj faze "Analiza zahtev" je zajem naročnikovih zahtev in želja. *Sprejemno testiranje* je torej zamišljeno kot nekakšno preverjanje če programska oprema ustreza vsem prej določenim zahtevam naročnika. Gre za nivo, kjer naročnik ali uporabnik s poglobljenim znanjem delovanja sistema preveri, ali le ta deluje ustrezno.

Faza "Načrtovanje" v razvoju programske opreme se posveča različnim komponentam, ki morajo skupaj delovati kot celota sistema, z vsemi že prej določenimi zahtevami. *Sistemsko testiranje* predvideva, da posamezne komponente delujejo in se usmerja v testiranje delovanja sistema kot celote. Odkritje nižjenivojskih napak na tem nivoju je lahko zelo drago, zato testiranje na tem nivoju ne izvaja naročnik ali programerji, temveč ločene testne ekipe.

*Integracijsko testiranje* preverja komunikacijo med moduli in usklajenost med vmesniki. Za uspešno testiranje se predvideva, da moduli delujejo brez napak. Najpogosteje so izvajalci isti kot pri sprejemem testiranju, torej testne ekipe.

Pri *Testiranju modulov* v fazi "Kodiranja" se usmerimo v samo strukturo in specifikacijo modulov. Modul si lahko lahko predstavljamo kot zbirko več enot (*ang. unit*), ki so zbrane v datoteki. V programskem jeziku `C#` se taka datoteka imenuje razred (*ang. class*). Cilj je testiranje modula v izolaciji in preverjanje usklajenosti enot v samem modulu. Večina podjetij za razvoj programske opreme prepušča skrb za testiranje modulov razvijalcem.

*Testiranje enot* spada na zadnji, "najnižji" nivo testiranja. Enoto si lahko predstavljamo kot spisek zaporednih programskih ukazov, ki jo program po potrebi kliče in potrebuje za delovanje. V programskem jeziku `C#` se enote imenujejo metode. S pravilnim in izčrpnim testiranjem dosežemo, da enote delujejo pravilno, kar je tudi podlaga za testiranje na ostalih nivojih. Ravno tako kot pri testiranju modulov, je za testiranje enot zadolžen programer.

Naj omenimo še *Regresijsko testiranje (ang. regression testing)*, ki se izvaja po vsaki spremembi v sistemu. Cilj regresijskega testiranje je zagotovitev, da sistem po spremembi pravilno deluje z vsemi funkcionalnostmi, ki jih je imel pred posodobitvijo [1].
Ker so napake, ki se odkrijejo šele na "višjih" nivojih testiranja lahko zelo drage, je zelo pomembno, da se testiranje izvaja vzporedno z razvojem projekta in ne na sredini ali pa šele na koncu.

## 2.3 Starejši delitvi testiranja

Pojem testiranje kot ga poznamo danes, je dobil pravi pomen šele v osemdesetih letih prejšnega stoletja, ko se je začelo razlikovati med razhroščevanjem in med samim testiranjem programske opreme v resničnem svetu [7]. Zaradi izredno hitrega razvoja tehnologije je v zadnjih dveh desetletjih prišlo do temeljnih sprememb glede samega pogleda na testiranje programske opreme. Eni izmed starejših delitev testiranja sta tudi t.i. metodi "črne štakle" in "bele škatle".

Metoda *črne škatle* testira funkcionalnost aplikacije brez vedenja o notranji strukturi oz. o delovanju programa [13]. Pri metodi *bele škatle* testiramo delovanje notranje strukture aplikacije. Ker poznamo strukturo in delovanje programa, lahko izberemo testne primere, ki bodo strukturo programa izčrpno preverili. Največkrat se uporablja na nivoju testiranja enot [12].

## 2.4 Pomembnost avtomatizacije testiranja

Testiranje programske opreme zahteva mišljenje, ki je drugačno od mišljenja razvijalca. Naloga programerja je napisati čim bolj varno, zanesljivo aplikacijo, medtem ko je naloga testerja odkriti šibke točke aplikacije. Čeprav se opis dela testerja zdi zanimiv, je testiranje programske opreme zelo drago in zahtevno, zato je eden izmed ciljev testiranja avtomatizacija testev, kar vodi k nižjim stroškom, zmanjšanju človeških napak in enostavnejšemu delu testerja.

Pomembnost avtomatizacije pride najbolj do izraza pri regresijskem testiranju, ki je lahko časovno zelo zahtevno. Poleg tega, da je ročni pristop k testiranju veliko bolj počasen, se izkaže, da je avtomatizacija bolj učinkovita pri iskanju napak. Ko so testi napisani, jih lahko poganjamo hitro in pogosto. Z vsako spremembo kode, pa obstaja tveganje, da določena funkcionalnost ne bo več delovala kot prej. Takrat nam avtomatizirani testi prihranijo veliko truda in časa [2].

## 2.5 Testiranje enot

Testiranje enot spada na zadnji, "najnižji" nivo, kar pa ne pomeni, da lahko to dejavnost zanemarimo. V resnici gre za prvi prikaz pravilnega delovanja sistema in je podlaga za nadaljna testiranja. Testiranja na višjih nivojih temeljijo na pravilnem in izčrpnem testiranju enot. V objektno usmerjenih programskih jezikih si lahko enote predstavljamo kot metode oz. razrede.

### 2.5.1 Osnovni koncepti

Za uspešno razumevanje diplomskega dela se bomo najprej seznanili z nekaterimi osnovnimi koncepti, ki se uporabljajo pri testiranju enot.

**Testni razred** (*ang. test class*): Testni razred vsebuje skupek testnih metod, ki testirajo isti objekt. Testni razred se od razreda, ki se uporablja pri objektno usmerjenih programskih jezikih razlikuje v tem, da pred samo definicjo razreda prednjači "atribut", ki prevajalniku pove, da je to testni razred. Atributi se od ogrodja do ogrodja razlikujejo, kar bomo tudi predstavili v nadaljevanju.

**Testna metoda** (*ang. test method*): Testna metoda je spisek ukazov, ki jih računalnik izvede. Programer si najprej testni primer izmisli, nato ga v programskem jeziku zapiše. S testnimi metodami testiramo pravilno delovanje osnovnih gradnikov aplikacije. Če se testna metoda neuspešno izvede, pomeni da imamo v našem programu napako.

**Trditev** (*ang. assertion*): Vska testna metoda ima lahko samo dve stanji: ali se test uspešno izvede (`true`), ali pa ne (`false`). Preverjanje stanja poteka na koncu testne metode s pomočjo metod ogrodja, katerim pravimo trditve. Trditve so najpomembnejše metode, saj z njimi preverimo rezultat napisanega testa. Njihova sintaksa se od ogrodja do ogrodja razlikuje, najbolj osnovna trditev pa preveri, ali sta dva podatka enaka (*ang. equality assert*).

**Atribut** (*ang. attribute)*: Atributi se pri testiranju enot uporabljajo za deklaracijo testnih razredov in testnih metod, recimo v NUnit ogrodju se testni razred defi-

nira z atributom `[TextFixture]`. Njihova uporaba pa še zdaleč ni omejena le na deklaracijo razredov in metod, saj nam omogočajo prilagoditev samega obnašanja testne metode oz. razreda. Velikokrat si želimo pred samo izvedbo testa ali po njem izvesti določene ukaze, kot so npr. inicializacija razreda, nastavitev spremenljivk ali uničenje objekta. S pomočjo atributov metode ustrezno označimo (v NUnit ogrodju se uporabljata atributa `SetUp` in `TearDown`) in jih zapišemo samo enkrat. S tem se izognemo nepotrebnemu ponavljanju kode, kar pripomore k jasnejši kodi. Obstajajo še drugi atributi, ki jih bomo bolj natančno predstavili v nadaljevanju.

### 2.5.2 Prednosti in slabosti

Tako testiranje enot kot ostale prakse za razvoj programske opreme imajo prednosti in tudi slabosti. Predstavimo najprej nekaj prednosti:

**Hitrejši razvoj**: Čeprav se zdi v nasprotju z logiko, nam lahko dobro napisani testi celo pospešijo pot do končnega produkta. To je mogoče ker napake v kodi odkrijemo prej, pa tudi zaradi lepše, bolje načrtovane kode, ki nam na dolgi rok omogoča večjo razširljivost.

**Lažje vzdrževanje**: Kodiranje programske opreme se ponavadi ne konča s predajo izdelka naročniku, saj predaji sledi še vzdrževanje. Vzdrževanje programske opreme je zapleten proces, kjer spreminjamo obstoječo kodo, dodajamo nove funkcionalnosti in odpravljamo napake. Z vsako spremembo kode pa obstaja možnost da določen modul ne bo več deloval kot prej. V primeru da smo se posluževali testiranja enot, lahko že napisane teste samo ponovno zaženemo in se prepričamo o pravilnem delovanju.

**Takojšnje povratne informacije**: Rezultati testiranja enot nam omogočajo takojšnje povratne informacije o trenutnem napredku. Sami testi enot pa nam lahko služijo tudi kot dokumentacija, ki bo bodočim razvijalcem v oporo.

**Boljša kakovost**: S pisanjem in izvajanjem testnih metod se povečuje kakovost in varnost napisane kode, kar vodi k boljšemu, kakovostnejšemu končnemu izdelku.

**Ponovna uporaba**: Testiranje enot lahko pomaga tudi pri ponovni uporabi (*ang. re-use*) kode. Že napisano kodo in teste enostavno prenesemo v trenutni projekt in nato spremenimo kodo, da se vsi testi uspešno izvedejo.

Kljub predstavljenim prednostim, ima testiranje enot tudi določene pomanjkljivosti.

**Dodatno delo**: Dodatno delo je eden izmed glavnih razlogov zakaj razvijalci niso naklonjeni pisanju testov za testiranje enot. Pisanje kode, ki bo samo preverila že napisano kodo se resda zdi časovno potratno, čeprav se na dolgi rok ne izkaže tako.

**Pomanjkljivo napisani testi**: Testi, ki so bili napisani površno oz. niso bili napisani z namenom odkrivanja napak, so neuporabni. S pisanjem neprimernih testov bomo dosegli le lažen občutek varnosti, da je programska oprema brez napak. Drži pa, da tudi z izčrpnim testiranjem enot vseh napak ne moremo odkriti.

### 2.5.3 Testiranje enot ali integracijsko testiranje

Zaradi manjših razlik med obema nivojema, veliko začetnikov oba nivoja enostavno enači. K zmedi pripomorejo tudi nekatera orodja, ki uporabljajo besedno zvezo "*unit test*" za teste, ki v resnici to niso. V tem podpoglavju bomo zato predstavili glavne razlike med obema nivojema.

Kot smo že omenili v podpoglavju 2.2, integracijsko testiranje preverja komunikacijo med moduli in usklajenost med vmesniki. Največja razlika med testiranjem enot in integracijskem testiranjem pa je v sami izolaciji metod. To v praksi pomeni, da testi za testiranje enot ne komunicirajo z zunanjimi viri, kot so baze (*ang. database)*, strežniki (*ang. server)*, spletne storitve (*ang. web service)*, poštni strežniki (*ang. mail server)* in datotečni sistem (*ang. file system)* [11]. V primeru, ko imamo v kodi zunanje vire, jih nadomestimo s tako imenovanimi imitiranimi objekti (*ang. mock objects*). Ravno zaradi izolacije metod se unit testi izvedejo veliko hitreje kot integracijski testi, kar pomeni, da jih lahko napišemo več in jih tudi pogosteje izvajamo.

# Poglavje 3

# Ogrodja

Ogrodje (*ang. framework)* si lahko predstavljamo kot nekakšno platformo za razvoj programske opreme. Ponuja nam osnovo, s pomočjo katere lahko razvijalci hitreje in varneje izdelajo programsko opremo. Ogrodje ponavadi vključuje predefinirane razrede in funkcije, ki lahko obdelajo vhodne podatke in komunicirajo s programsko opremo. S tem pohitrimo razvoj, saj razvijalcem ni potrebno skrbeti za infrastrukturo aplikacije [9].
Ogrodja nam tako poenostavljajo določena opravila, v našem primeru pa nam omogočajo tudi nadzorovano okolje za pisanje in izvajanje testov.
Izbira pravega ogrodja ni vedno lahka, saj na odločitev vpliva veliko dejavnikov. Izmed vseh ogrodij ki obstajajo za programski jezik `C#`, smo izbrali tri, ki se po našem mnenju trenutno največ uporabljajo pri razvoju programske opreme in so med razvijalci najbolj priljubljena. To so ogrodja NUnit, xUnit in MSTest.

## 3.1 Ogrodje NUnit

### 3.1.1 Zgodovina in razvoj

NUnit je ogrodje za testiranje enot, ki je namenjeno vsem .NET programskim jezikom [8]. Razvito je bilo po zgledu ogrodja JUnit, ki se uporablja za testiranje enot v programskem jeziku Java. Trenutna produkcijska verzija (v času pisanja je to 2.6) je sedma večja izdaja tega ogrodja. Orodje je v celoti napisano v programskem jeziku `C#`, sama programska koda pa je bila zaradi implementacije novih

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using NUnit.Framework;
7
8  namespace NUnit
9  {
10     [TestFixture]
11     class BankAccountTest
12     {
13         [Test]
14         public void depositMoney()
15         {
16             BankAccount account = new BankAccount();
17             account.deposit(50);
18             double expected = 50;
19             Assert.AreEqual(expected, account.total);
20         }
21
22         [Test]
23         public void withdrawMoney()
24         {
25             BankAccount account = new BankAccount();
26             account.deposit(200);
27             account.withdraw(40);
28             double expected = 160;
29             Assert.AreEqual(expected, account.total);
30         }
31     }
32 }
```

Slika 3.1: Testni razred z dvema testnima metodama v ogrodju NUnit

.NET programskih funkcij tudi ustrezno spremenjena.

### 3.1.2 Osnovni gradniki

**Testni razred in testna metoda**: Pri vseh ogrodjih za testiranje enot potrebujemo nekakšne oznake, ki prevajalniku povedo, da v izbranem primeru ne gre za navaden razred ali metodo, temveč za testni razred oz. testno metodo. Za deklaracijo testnega razreda v ogrodju NUnit se tako uporabi atribut `[TestFixture]`, za testno metodo pa enostavno `[Test]`. Primer preprostega testnega razreda si lahko ogledamo na sliki 3.1.

**Trditvene metode**: Ogrodje NUnit razpolaga z množico trditvenih metod, katere se nahajajo v razredu `Assert`. Od različice 2.4 naprej se trditvene metode delijo na dva modela: t.i. "klasični model" (*ang. classic model*) in "model z omejitvami" (*ang. constraint model*). Pri klasičnem modelu se za vsako trditev uporablja različna metoda razreda `Assert` (`Assert.AreEqual`, `Assert.AreNotEqual`,

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using NUnit.Framework;
7
8  namespace NUnit
9  {
10     [TestFixture]
11     class BankAccountTestSetUp
12     {
13         private BankAccount account;
14         [SetUp]
15         public void init()
16         {
17             account = new BankAccount();
18         }
19         [Test]
20         public void depositMoney()
21         {
22             account.deposit(50);
23             double expected = 50;
24             Assert.AreEqual(expected, account.total);
25         }
26
27         [Test]
28         public void withdrawMoney()
29         {
30             account.deposit(200);
31             account.withdraw(40);
32             double expected = 160;
33             Assert.AreEqual(expected, account.total);
34         }
35     }
36 }
```

Slika 3.2: Uporaba atributa `SetUp` za inicializacijo razreda `BankAccount`

`Assert.IsTrue`, `Assert.False`,...). Drugi model pa za vse trditve uporablja eno metodo razreda `Assert` (`Assert.That`). Trenutno sta oba modela podprta, saj večina razvijalcev raje uporablja "klasičen model" [8].

**Atributi**: Atribute smo delno že spoznali pri deklaraciji testnih razredov in testnih metod. V primeru na sliki 3.1 se koda za inicializacijo razreda `BankAccount` ponovi na začetku vsake testne metode. Da pripomoremo k lepši in berljivejši kodi, lahko uporabimo atribut `[SetUp]`, kot je to prikazano na sliki 3.2. Naj omenimo še nekatere pomembnejše atribute, kot so: `Exception` (pričakujemo, da bo testna metoda vrnila izjemo), `Ignore` (test se trenutno ne bo izvedel), `Maxtime` (maksimalen čas v milisekundah, v katerem se mora test izvesti), `Repeat` (testna metoda se bo izvedla večkrat), `TearDown` (izvede se po vsaki testni metodi) in seveda atribut `Test`, ki označuje testno metodo.

**Izvajanje**: Ogrodje nam omogoča poganjanje testnih metod na tri različne načine:

- preko ukazne vrstice;
- s pomočjo NUnit grafičnega vmesnika;
- neposredno iz nekaterih razvojnih okolij, kot je npr. Visual Studio.

Pri ogrodju NUnit nam je zelo všeč njihova spletna stran, saj na njej najdemo obširno dokumentacijo z začetnimi napotki, navodili za namestitev, različnimi primeri kode in tudi možnost prenosa celotne dokumentacije na računalnik. Največja pomankljivost se po našem mnenju odraža v samem razvoju ogrodja, ki je v zadnjih letih malenkost zastal. Nove verzije izhajajo poredkoma in tudi ne vsebujejo novih funkcionalnosti, ampak večinoma samo manjše popravke.

## 3.2 Ogrodje xUnit

### 3.2.1 Zgodovina in razvoj

xUnit je zastonjsko, odprtokodno (*ang. open source*) orodje za testiranje enot, katerega avtor je tudi izumitelj ogrodja NUnit v2. Je najnovejša tehnologija za testiranje enot v programskih jezikih: `C#`, `F#`, VB.NET in nekateri drugi. Nastalo je zaradi določenih pomankljivosti ogrodja NUnit: slaba izolacija testnih razredov in preveliko število nepotrebnih atributov [16].

### 3.2.2 Osnovni gradniki

**Testni razred in testna metoda**: Za razliko od ogrodja NUnit, deklaracija testnega razreda tukaj ni potrebna. Pomembno je samo, da je razred javen (`public class`), da lahko xUnit ustrezno zazna teste. Pri testni metodi je sintaksa drugačna, saj za označitev le te uporabimo atribut `[Fact]` (slika 3.3).

**Trditvene metode**: xUnit je v primerjavi z ogrodjem NUnit pri trditvenih metodah odstranil večino pojavitev besed `Are` in `Is`. Tako v ogrodju xUnit ne bomo našli trditvenih metod, kot so `AreEqual` ali `AreNotSame`, temveč enostavno `Equal` in `NotSame`.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using Xunit;
7
8  namespace xUnitClass
9  {
10     public class BankAccountTest
11     {
12         [Fact]
13         public void depositMoney()
14         {
15             BankAccount account = new BankAccount();
16             account.deposit(50);
17             double expected = 50;
18             Assert.Equal(expected, account.total);
19         }
20
21         [Fact]
22         public void withdrawMoney()
23         {
24             BankAccount account = new BankAccount();
25             account.deposit(200);
26             account.withdraw(40);
27             double expected = 160;
28             Assert.Equal(expected, account.total);
29         }
30     }
31 }
```

Slika 3.3: Testni razred z dvema testnima metodama v ogrodju xUnit

**Atributi**: Zaradi velikega števila atributov v ogrodju NUnit so se razvijalci xUnita odločili za drugačen pristop, zato so odstranili oz. modificirali nekatere atribute. Tako bomo tukaj zaman iskali atribute: `[TestFixture]`, `[Setup]`, `[TearDown]`, `[TestFixtureSetup]`, `[TestFixtureTearDown]`, za začasno neizvajanje testa pa je potrebno uporabiti ključno besedo `Skip`.

**Izvajanje**: Ogrodje nam omogoča poganjanje testnih metod na štiri različne načine:

- preko ukazne vrstice;
- MSBuild (orodje, ki nam s pomočjo XML datotek omogoča izdelavo .NET projektov in aplikacij brez namestitve Visual Studio razvojnega okolja);
- s pomočjo xUnit grafičnega vmesnika;
- neposredno iz Visual Studio razvojnega okolja.

Največja prednost ogrodja xUnit pred ostalimi je v sami zasnovi. Zaradi velike količine atributov in trditev se nam lahko zazdi, da smo se primorani naučiti nov programski jezik. Tega so se razvijalci zavedali, zato so odstranili nekatere atribute in trditve, poenostavili sintakso in omogočili zaznavo testnih metod brez deklaracije testnih razredov.

## 3.3 Visual Studio testno ogrodje (MSTest)

### 3.3.1 Zgodovina in razvoj

Microsoft je razvojno okolje Visual Studio razvil z namenom lažjega, bolj učinkovitega in hitrejšega pisanja kode. Ker pa Visual Studio testiranja enot v začetnih izdajah ni podpiral, so se razvila ogrodja, ki so razvijalcem to omogočala. Zaradi vse večjega zavedanja o pomembnosti učinkovito stestirane programske opreme, so se pri Microsoftu odločili za vključitev testnega ogrodja neposredno v razvojno okolje Visual Studio. Zaradi vključitve, razvijalcem ni potrebno poseči po alternativnih ogrodjih.

### 3.3.2 Osnovni gradniki

**Testni razred in testna metoda**: Enako kot pri ogrodjih NUnit in xUnit se za označitev tesnega razreda in testnih metod tudi tukaj uporablja atribute. Testni razred označimo z atributom `[TestClass]`, testne metode pa z `[TestMethod]` (slika 3.4). Edina večja razlika med prejšnjima ogrodjima je v tem, da za uporabo integriranega testnega ogrodja v Visual Studiu ne potrebujemo dodatne programske opreme.

**Trditvene metode**: Večina trditvenih metod je tudi tukaj zbranih v razredu `Assert`. Na razpolago pa imamo več različnih vrst trditev, ki se delijo v naslednje razrede:

- `Assert`: Assert razred vsebuje množico metod: `AreEqual`, `AreNotEqual`, `AreSame`, `AreNotSame`, `IsFalse`, `IsTrue` in ostale.
- `CollectionAssert`: Uporablja se za primerjavo zbirk (*ang. collection*).

```
1  using System;
2  using Microsoft.VisualStudio.TestTools.UnitTesting;
3  using VisualStudioTesting;
4
5  namespace BankAccountTest
6  {
7      [TestClass]
8      public class BankAccountTest
9      {
10         [TestMethod]
11         public void depositMoney()
12         {
13             BankAccount account = new BankAccount();
14             account.deposit(50);
15             double expected = 50;
16             Assert.AreEqual(expected, account.total);
17         }
18
19         [TestMethod]
20         public void withdrawMoney()
21         {
22             BankAccount account = new BankAccount();
23             account.deposit(200);
24             account.withdraw(40);
25             double expected = 160;
26             Assert.AreEqual(expected, account.total);
27         }
28     }
29 }
```

Slika 3.4: Preprost testni razred z dvema testnima metodama v ogrodju MSTest

- `StringAssert`: Razred nam ponuja dodatne metode za primerjavo nizov [14].

**Atributi**: V primerjavi s prejšnjima ogrodjema imamo tukaj nekaj manj atributov. Še vedno najdemo atribute, ki se izvedejo takoj po inicializaciji ali uničenju razreda (`ClassInitializeAttribute` in `ClassCleanupAttribute`), ravno tako obstajajo atributi za manipulacijo izvajanja same testne metode (`TestInitializeAttribute` in `TestCleanupAttribute`).

**Izvajanje**: Ogrodje nam omogoča poganjanje testnih metod na dva različna načina:

- neposredno iz Visual Studio razvojnega okolja;
- preko ukazne vrstice (MSTest.exe).

Vključitev ogrodja za testiranje enot v razvojno okolje Visual Studio odstrani potrebo po nameščanju dodatne programske opreme. Zagon testov je znotraj okolja enostaven, njihov prikaz pa pregleden. Veliko slabost smo opazili pri sami dokumentaciji, saj je uradna spletna stran izredno nepregledna, število člankov in vodičev na internetu pa je zaradi slabše popularnosti nizko, kar lahko začetnikom povzroča nemalo težav.

# Poglavje 4

# Primerjava ogrodij

Poglavje je v celoti namenjeno primerjavi ogrodij NUnit, xUnit in Visual Studio testnem ogrodju. Za primerjavo smo uporabili najnovejše stabilne verzije orodij (NUnit 2.6.4, xUnit 1.9.2 in razvojno okolje Visual Studio Ultimate 2012) z najnovejšimi popravki.

Glavni cilj poglavja je čimbolj učinkovita in objektivna primerjava ter analiza izbranih orodij. Zaradi velikega števila različnih ogrodij za testiranje enot so lahko razvijalci velikokrat v dilemi katero orodje sploh izbrati. Zato smo primerjavo poizkušali narediti skozi oči razvijalca, ki ima od ogrodja določene zahteve.

## 4.1 Kriteriji primerjave

Za primerjavo smo izbrali kriterije, ki se v veliki meri posvečajo na vidik uporabnosti ogrodja. To so:

- obstoj dokumentacije,
- popularnost,
- integracija z razvojnim okoljem Visual Studio,
- izdelava poročil,
- možnosti konfiguracije testov,

- trditvene metode,
- način izvajanja testiranja,
- razvoj ogrodja,
- uporabnost v praksi.

### 4.1.1 Obstoj dokumentacije

Zaradi velike količine atributov in trditvenih metod je pri vsakem ogrodju zelo pomemben obstoj uradne dokumentacije kot tudi obstoj skupnosti, ki nam lahko pomaga ob začetnih težavah.

**NUnit**
Prva verzija ogrodja NUnit sega v leto 2002, kar ga postavlja med najstarejša ogrodja, ki jih v diplomski nalogi primerjamo. Zaradi daljše zgodovine lahko na internetu najdemo veliko število člankov in vodičev, ki nam pomagajo pri začetnih korakih. Obstaja pa tudi zelo velika spletna skupnost. Ne smemo pozabiti tudi na uradno spletno stran, na kateri imamo na razpolago pregledno dokumentacijo za vsako verzijo ogrodja.

**xUnit**
Pet let po izidu ogrodja NUnit (leta 2007) se je na spletu pojavilo novo ogrodje z imenom xUnit, katerega avtor je tudi izumitelj ogrodja NUnit v2. Glede popularnosti se orodje seveda ne more primerjati z ogrodjem NUnit, saj obstaja manj časa. Dokumentacijo za lažji začetek najdemo na spletni strani, kjer dobimo tudi odgovore na najpogosteje zastavljena vprašanja, ki se tičejo namestitve, integracije z razvojnimi okolji in predstavitev različnih načinov za poganjanje testov.

**MSTest**
Visual Studio testno ogrodje (MSTest) je odgovor Microsofta na množico različnih orodij za testiranje enot. Ogrodje je definirano v `dll` (*ang. Dynamic-link library)* datoteki z imenom `Microsoft.VisualStudio.QualityTools.UnitTestFramework`. Dodatne informacije o ogrodju lahko pridobimo na Microsoftovi MSDN spletni

strani, ki je po našem mnenju zaradi velike količine podatkov o različnih tehnologijah izredno nepregledna in nerazumljiva za začetnika.

**NUnit**:

+ veliko število člankov in vodičev
+ velika spletna skupnost
+ pregledna dokumentacija

**xUnit**:

+ pregledna dokumentacija
+ solidno število člankov in vodičev

**MSTest**:

- dokumentacija obstaja, vendar je nepregledna
- majhno število člankov in vodičev

| NUnit | xUnit | MSTest |
|---|---|---|
| 5 | 4 | 2 |

Tabela 4.1: Ocene ogrodij za kriterij "obstoj dokumentacije"

### 4.1.2 Popularnost

Popularnost ogrodja je lahko zelo pomemben dejavnik. Poznavanje popularnega ogrodja za testiranje enot lahko tudi pomaga pri bodoči zaposlitvi.
Eden izmed načinov za preverjanje popularnosti ogrodja je preprosto spletno iskanje. Glede na število zadetkov, ki nam jih spletni iskalniki vrnejo, lahko posredno sklepamo o popularnosti ogrodja. Tabela 4.2 prikazuje približno število zadetkov v spletnih iskalnikih Google [5], Bing [3] in Yahoo [17]. V vseh treh iskalnikih smo vnesli iskalne nize: "nunit", "xunit" in "mstest".

**NUnit**:

+ najbolj popularno ogrodje (glede na rezultate spletnih iskalnikov)

| Spletni iskalnik | `NUnit` | `xUnit` | `MSTest` |
|---|---|---|---|
| `Google` | 1.640.000 | 834.000 | 752.000 |
| `Bing` | 658.000 | 158.000 | 415.000 |
| `Yahoo` | 521.000 | 129.000 | 400.000 |

Tabela 4.2: Približno število zadetkov v najpopularnejših spletnih iskalnikih na dan 3.5.2014

**xUnit**:

+ pridobiva na popularnosti

- najmanj popularno ogrodje glede na iskalnika Bing in Yahoo

**MSTest**:

+ pridobiva na popularnosti

| **NUnit** | **xUnit** | **MSTest** |
|---|---|---|
| 5 | 3 | 4 |

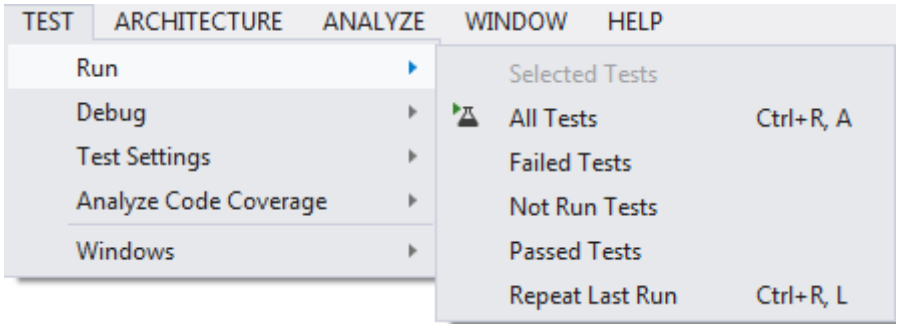
Tabela 4.3: Ocene ogrodij za kriterij "popularnost"

### 4.1.3 Integracija z razvojnim okoljem Visual Studio

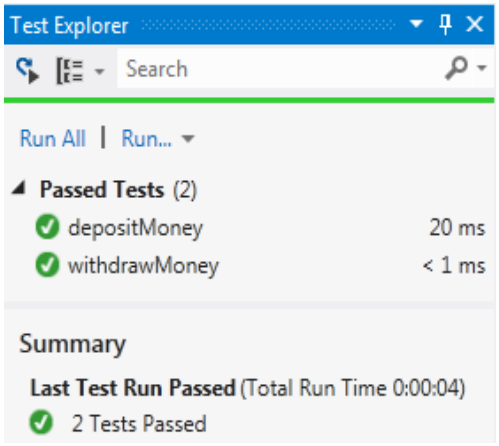
Integrirano razvojno okolje (*ang. integrated development environment*) je programska aplikacija, ki razvijalcem ponuja množico pripomočkov in orodij za lažje delo. Večina razvojnih okolij vsebuje elemente, kot so: urejevalnik kode, razhroščevalnik, prevajalnik in celo pametno dopolnjevanje kode [6].
Situacija je podobna pri testiranju. Tester si želi, da lahko testiranje enot enostavno izvede kar iz razvojnega okolja. Zato smo si za kriterij želeli preveriti, ali nam izbrana ogrodja omogočajo integracijo z najpopularnejšim razvojnim okoljem za .NET programske jezike - razvojno okolje Visual Studio.

**NUnit**

Za ogrodje NUnit obstaja vtičnik, ki pa ni privzeto nameščen v Visual Studio,



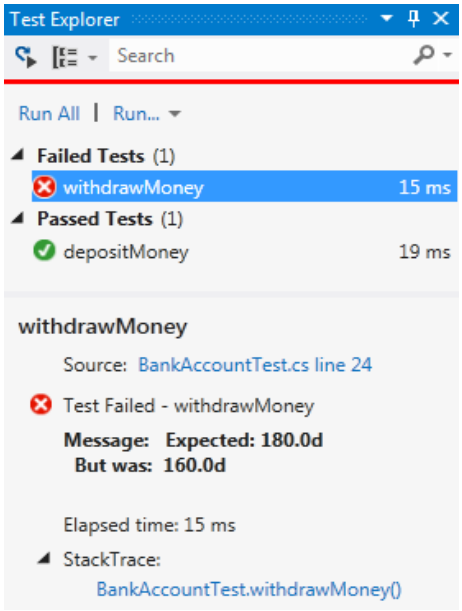

Slika 4.1: Zagon testov v razvojnem okolju Visual Studio




Slika 4.2: Prikaz rezultatov v oknu `Test Explorer`

kar pomeni da ga je najprej potrebno namestiti. Namestitev lahko zaženemo neposredno iz Visual Studio razvojnega okolja s pomočjo menija `Extensions and Updates`. Ko se prenos uspešno zaključi, lahko teste izvedemo s klikom na menu `Test` → `Run` → `All Tests` ali s pomočjo bližnjice `Ctrl+R, A`. (slika 4.1)

Po končani izvedbi testov se nam prikaže okno `Test Explorer`, ki nam omogoča hiter in enostaven pregled nad vsemi testi. Razberemo lahko število vseh testov, ali se je test uspešno izvedel in njegovo trajanje. (slika 4.2) V primeru, da se test ni uspešno izvedel, dobimo kratek opis napake. (slika 4.3)




Slika 4.3: Neuspešen test s kratkim opisom napake

**xUnit**

Pri ogrodju xUnit je situacija zelo podobna kot pri prejšnjem orodju. Ker ogrodje ni privzeto nameščeno v razvojno okolje, ga je najprej potrebno namestiti. Postopek se tudi tukaj izvede preko menija `Extensions and Updates`. Z bližnjico `Ctrl+R, A` poženemo vse teste v testnem razredu, lahko pa v oknu `Test Explorer` izberemo testno metodo, ki jo želimo izvesti. Po končanem testiranju se nam odpre enako okno kot pri ogrodju NUnit (`Test Explorer`), ki nam prikaže uspešne, neuspešne teste in čas trajanja le teh.

**MSTest**

Microsoftovo testno ogrodje MSTest je za razliko od prej obravnavanih ogrodij že nameščeno v razvojno okolje Visual Studio. Drugih razlik ni, saj teste tudi tukaj poženemo z identično kombinacijo tipk. Ravno tako se za prikaz stanja testov uporablja že prej omenjeno okno (`Test Explorer`).

**NUnit**:

+ enostavna namestitev
- ogrodje je potrebno dodatno namestiti

**xUnit**:

+ enostavna namestitev

- ogrodje je potrebno dodatno namestiti

**MSTest**:

+ ogrodje je že nameščeno v razvojno okolje Visual Studio

| NUnit | xUnit | MSTest |
|---|---|---|
| 4 | 4 | 5 |

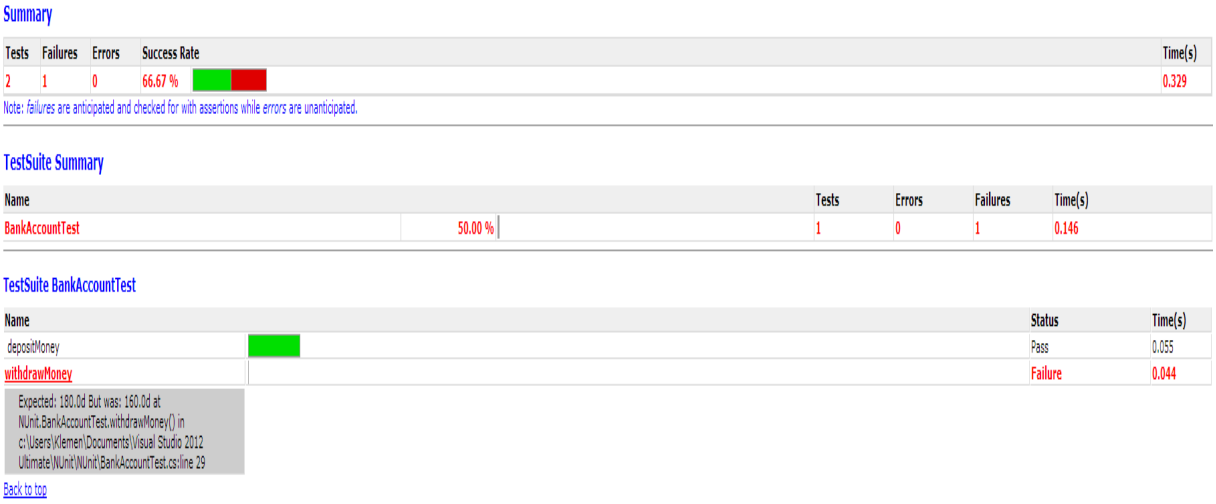
Tabela 4.4: Ocene ogrodij za kriterij "integracija z razvojnim okoljem Visual Studio"

### 4.1.4 Izdelava poročil

Testiranje je aktivnost, ki naj bi se praviloma izvajala vzporedno s samim razvojem projekta. Zato je za razvijalca velikokrat dovolj, da se rezultati testiranja enot prikažejo kar neposredno v razvojnem okolju. Velikokrat pa tak prikaz ni ustrezen (recimo ko vodja ekipe želi spremljati razvoj projekta in o napredku poročati vodstvu), kar pomeni da je rezultate testov treba prikazati na "lepši", grafični način oziroma izdelati poročilo. Poleg že omenjene vloge so poročila lahko uporabna tudi kot dokumentacija za ostale razvijalce. Vizualni prikaz nam omogoča tudi enostaven in hiter pregled trenutnega stanja.
Pri kriteriju bomo preverili ali nam ogrodje omogoča generiranje poročil znotraj samega ogrodja ali pa s pomočjo ostalih orodij. Poročilo mora biti jasno in pregledno.

**NUnit**

Znotraj samega ogrodja NUnit izdelava poročil ni mogoča, kar pomeni, da je za tovrstno funkcionalnost treba poseči po alternativnih orodjih. Eno izmed takih je orodje `NUnit2Report`. NUnit nam po izvedenih testih zgenerira `XML` datoteko z imenom `TestResult.xml`. `NUnit2Report` je konzolna aplikacija, ki nam `XML` datoteko z rezultati testiranja (`TestResult.xml`) s pomočjo XSLT tranformacij pretvori v pregledno poročilo v `HTML` formatu. (slika 4.4).




Slika 4.4: Poročilo, zgenerirano z orodjem `NUnit2Report`

### xUnit

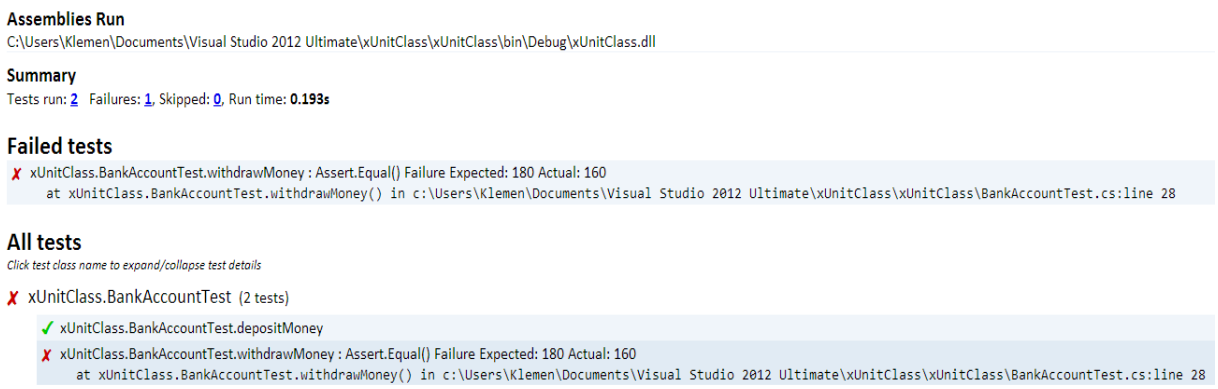
Orodje xUnit lahko poganjamo na več različnih načinov. Najpogosteje se uporablja grafični vmesnik, za generiranje poročil pa je najbolj uporaben način z uporabo ukazne vrstice. Ukazna vrstica nam ponuja dodatne možnosti, s katerimi lahko določimo izhodni format datoteke. Tako lahko rezultate testiranja pridobimo v:

- xUnit `XML` obliki;
- `HTML` formatu;
- NUnit `XML` obliki.

Če izberemo NUnit `XML` format, potem lahko izbrano datoteko enostavno pretvorimo v HTML format z že prej uporabljenim orodjem `NUnit2Report`. Za razliko od NUnit ogrodja nam xUnit omogoča izdelavo poročil v `HTML` formatu brez dodatnih orodij. (slika 4.5)

### MSTest

Tudi pri tem ogrodju je izdelava poročil zelo podobna prejšnjim. Ogrodje nam omogoča izdelavo poročila le preko ukazne vrstice, vendar izhodnega formata ne moremo določiti. Tako ob koncu testiranja dobimo datoteko s končnico `TRX`




Slika 4.5: Zgeneriano poročilo v xUnit ogrodju (`HTML` format)

(*Visual Studio Test Results File*), ki pa je v resnici `XML` datoteka. Za izdelavo poročila potrebujemo alternativno orodje, ki nam datoteko pretvori v drugi format. Eno izmed takih je orodje `trx2html`, ki nam trx datoteko pretvori v poročilo v `HTML` formatu. (slika 4.6)

**NUnit**:

- za izdelavo poročila je potrebno alternativno orodje

**xUnit**:

+ ogrodje nam privzeto omogoča izdelavo poročil

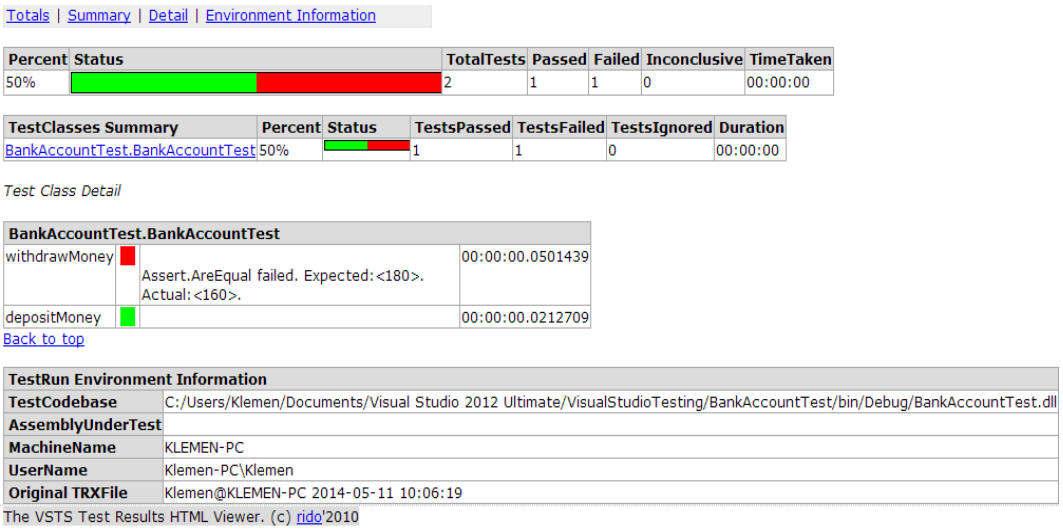
+ možnost izbire različnih izhodnih formatov (tudi za ogrodje NUnit)

**MSTest**:

- za izdelavo poročila je potrebno alternativno orodje

| NUnit | xUnit | MSTest |
|---|---|---|
| 3 | 5 | 3 |

Tabela 4.5: Ocene ogrodij za kriterij “izdelava poročil“



Totals | Summary | Detail | Environment Information

| Percent | Status | TotalTests | Passed | Failed | Inconclusive | TimeTaken |
|---|---|---|---|---|---|---|
| 50% | | 2 | 1 | 1 | 0 | 00:00:00 |

| TestClasses Summary | Percent | Status | TestsPassed | TestsFailed | TestsIgnored | Duration |
|---|---|---|---|---|---|---|
| BankAccountTest.BankAccountTest | 50% | | 1 | 1 | 0 | 00:00:00 |

*Test Class Detail*

| BankAccountTest.BankAccountTest | | | |
|---|---|---|---|
| withdrawMoney | | Assert.AreEqual failed. Expected:<180>. Actual:<160>. | 00:00:00.0501439 |
| depositMoney | | | 00:00:00.0212709 |

Back to top

| TestRun Environment Information | |
|---|---|
| TestCodebase | C:/Users/Klemen/Documents/Visual Studio 2012 Ultimate/VisualStudioTesting/BankAccountTest/bin/Debug/BankAccountTest.dll |
| AssemblyUnderTest | |
| MachineName | KLEMEN-PC |
| UserName | Klemen-PC\Klemen |
| Original TRXFile | Klemen@KLEMEN-PC 2014-05-11 10:06:19 |

The VSTS Test Results HTML Viewer. (c) rido'2010

Slika 4.6: Končni izgled poročila s pomočja orodja `trx2html`

### 4.1.5 Možnosti konfiguracije testov

Vsa obravnavana testna ogrodja nam prilagajanje testov omogočajo z atributi. Atribute smo že srečali pri deklaraciji testnih razredov in testnih metod. Gre za posebne oznake, s katerimi testnim razredom dodajamo metapodatke, ki ogrodju podajo dodatne informacije za izvedbo testov. Tako lahko ogrodju povemo, da pri določeni testni metodi pričakujemo izjemo (*ang.* *exception*), določimo najdaljši čas izvajanja metode, po možnosti pa tudi začasno preprečimo njeno izvajanje.

**NUnit**

Atribute lahko enostavno prepoznamo po njihovi sintaksi, saj so vedno napisani med dvema oglatima oklepajema (npr. atribut `[Test]`), ki označuje testno metodo.

Ker izvajanja programa zaradi zunanjih dejavnikov (kot so npr. uporabniki) ne moremo predvideti, moramo pri testnih metodah testirati tudi izjeme. V ogrodju NUnit lahko za ta namen uporabimo atribut `[ExpectedException]`. Funkcija lahko prejme več parametrov, lahko pa tudi nobenega. Za izčrpnejše testiranje moramo seveda uporabiti več parametrov. Slika 4.7 prikazuje testno metodo, ki bo označena kot "Uspešna" samo v primeru, ko bo vrnila izjemo tipa

`ArgumentException` s sporočilom, ki bo vsebovalo besedo “unspecified“.

```
[ExpectedException("System.ArgumentException", ExpectedMessage = "unspecified", MatchType = MessageMatch.Contains)]
public void testMethod()
{

}
```

Slika 4.7: Primer testne metode v ogrodju NUnit, kjer pričakujemo izjemo

Včasih se pojavi potreba, da neke metode trenutno ne želimo stestirati, saj mogoče ne več obstaja ali pa koda zanjo ni dokončana. Takrat lahko uporabimo atribut `Ignore`, ki nam tako označeno metodo začasno preskoči.

**xUnit**

Ogrodje xUnit za označitev testnih metod uporablja atribut `[Fact]`, za testni razred pa atribut ni potreben, saj orodje samo poišče vse testne metode v vseh javnih razredih.

Zaradi težav, kot so:

- natančna vrstica kode, ki je vrnila izjemo ni znana,
- nimamo možnosti natančne analize izjeme,

ki se lahko pojavijo pri uporabi atributa `[ExpectedException]` so se razvijalci odločili za ukinitev tega v prid funkciji `Assert.Throws` (slika 4.8). Za ignoriranje testne metode pa uporabimo atribut `[Fact(Skip=‘‘razlog‘‘)]`.

```
[Fact]
public void argumentException()
{
    BankAccount account = new BankAccount();
    Assert.Throws<System.ArgumentException>(() => account.deposit(-100));
}
```

Slika 4.8: Uporaba funkcije Assert.Throws za testiranje izjem v ogrodju xUnit

**MSTest**
Tudi v MSTest ogrodju se za konfiguracijo testov uporablja atribute, zato velikih razlik tukaj ne bomo našli. Za deklaracijo testnega razreda uporabimo atribut `[TestClass]`, za testno metodo pa `[TestMethod]`. Sintaksa za testiranje izjem in ignoriranje testnih metod je identična kot pri ogrodju NUnit (atributa `[ExpectedException]` in `[Ignore]`).

Večjih razlik pri trenutnem kriteriju nismo našli. Smo pa opazili, da so se razvijalci ogrodja xUnit naučili iz napak, ki so jih naredili pri NUnit orodju, kar se lepo opazi pri atributih, saj so po našem mnenju bolj dodelani in premišljeni.

**NUnit**:
- testni razred je potrebno deklarirati

**xUnit**:
+ deklaracija testnih razredov ni potrebna
+ bolj dodelana in učinkovita uporaba atributov

**MSTest**:
- testni razred je potrebno deklarirati

| NUnit | xUnit | MSTest |
|---|---|---|
| 4 | 5 | 4 |

Tabela 4.6: Ocene ogrodij za kriterij "možnosti konfiguracije testov"

### 4.1.6 Trditvene metode

Trditve so metode, ki predstavljajo najpomembnejši del ogrodja za testiranje enot. V vsaki testni metodi se ob koncu njenega izvajanja izvede preverjanje pogoja, ki test označi kot uspešen ali neuspešen. To preverjanje poteka s pomočjo trditvenih metod.

**NUnit**

Ogrodje NUnit za trditve uporablja dva modela: "klasični model" in "model z omejitvami". Trditvene metode v klasičnem modelu zaradi preglednosti delimo v več skupin:

- Trditve enakosti (`Assert.AreEqual`, `Assert.AreNotEqual`);
- Trditve identitete (`Assert.AreSame`, `Assert.AreNotSame`, `Assert.Contains`);
- Trditve primerjav (`Assert.Greater`, `Assert.GreaterOrEqual`, `Assert.Less`,...);
- Trditve tipov (`Assert.IsInstanceOfType`, `AssertInstanceOf` <T>);
- Testiranje pogojev (`Assert.IsTrue`, `Assert.IsFalse`, `Assert.IsNull`, `Assert.IsNotNull`,...);
- Pomožne metode (`Assert.Pass`, `Assert.Fail`, `Assert.Inconclusive`).

Novejši model (model z omejitvami) zaradi spremenjene sintakse ni bil pretirano sprejet, zato lahko razvijalci še vedno uporabljamo oba. Razliko med obema modeloma v primerjavi dveh vrednosti si lahko ogledamo na sliki 4.9. Ker je novejši model zelo obširen, bomo našteli samo najpomemnejše skupine elementov:

- Omejitev enakosti (omogoča nam primerjavo dveh vrednosti);
- Omejitev pogoja (omogoča nam primerjavo različnih pogojev: `Is.True`, `Is.Null`, `Is.Null`, `Is.Unique`);
- Omejitev primerjav (uporablja se pri primerjanju velikosti dveh številk: `IsGreaterThan`, `Is.GreaterThanOrEqualTo`,...);
- Omejitev tipov (preverimo ali je objekt pravega tipa - niz, številka,...);
- Omejitev niza (vsebuje dodatne metode za primerjavo nizov).

**xUnit**

Ob primerjavi ogrodja xUnit z ostalimi bomo najprej opazili spremembo pri poimenovanju metod. Le-te se še vedno nahajajo v razredu `Assert`, vendar so programerji odstranili večino pojavitev besed `Are` in `Is`. Tako metod, kot so `AreEqual`,

`AreSame` ne bomo našli, saj so jih nadomestile metode `Equal` in `Same`. Dodanih pa je bilo tudi nekaj trditev, ki jih bomo v ogrodjema NUnit in MSTest zaman iskali. To so:

- `InRange` (preverimo, ali je vrednost v izbranem območju);
- `NotInRange` (preverimo, ali vrednost ni v izbranem območju);
- `Throws` (zamenjava za atribut `[Exception]`).

**MSTest**
Večina trditvenih metod se tudi tukaj nahaja v razredu `Assert`, vendar nam ogrodje ponuja dodatne trditvene razrede, ki nam tovrstno funkcionalnost razširijo. Tako lahko za primerjavo zbirk uporabimo razred `CollectionAssert`, za primerjavo nizov pa `StringAssert`.

Pri sledečem kriteriju so se razlike izkazale za izjemno majhne, kar je z vidika uporabnika odličen podatek, saj v primeru prehoda na drugo ogrodje pomeni manj dodatnega prebiranja dokumentacije.

| NUnit | xUnit | MSTest |
|---|---|---|
| 4 | 4 | 4 |

Tabela 4.7: Ocene ogrodij za kriterij "trditvene metode"

### 4.1.7 Način izvajanja testiranja

Glavna komponenta vsakega testnega ogrodja je testni razred. Izvajanje testnega razreda je določeno z njegovim življenjskim ciklom. Pri tem kriteriju se bomo posvetili življenjskim ciklom izbranih ogrodij in predstavili težave, ki se lahko pojavijo v primeru slabega razumevanja le-teh.

**NUnit**
Pri ogrodju NUnit se vse testne metode izvajajo v isti instanci testnega razreda,

```
[Test]
public void withdrawMoney()
{
    BankAccount account = new BankAccount();
    account.deposit(200);
    account.withdraw(40);
    double expected = 160;

    //classic model
    Assert.AreEqual(expected, account.total);

    //constraint model
    Assert.That(expected, Is.EqualTo(account.total));
}
```

Slika 4.9: Testna metoda z dvema trditvama v ogrodju NUnit - klasični model in model z omejitvami

kar pomeni da je razvijalec dolžan poskrbeti za izoliranost testnih metod. To lahko storimo z uporabo že prej omenjenega atributa `SetUp`, ki se izvede pred vsakim testom.
Na sliki 4.10 lahko vidimo testni razred z dvema testnima metodama, kjer se pod deklaracijo testnega razreda najprej zgodi incializacija razreda `BankAccount`, ki vsebuje metode za delo z bančnim računom (`depositMoney`, `withdrawMoney`) in spremenljivko za skupni znesek na računu (`total`). Ker ogrodje NUnit vse testne metode izvaja v isti instanci testnega razreda, se bo incializacija razreda `BankAccount` zgodila samo enkrat, kar bo privedlo do napačnega rezultata pri drugem testu (slika 4.11), saj bo stanje na računu po prvi testni metodi 50 in ne 0, kot smo predvidevali med pisanjem drugega testa.
Težavo lahko rešimo tako, da napišemo metodo ki bo poskrbela za inicializacijo razreda `BankAccount` in jo označimo z atributom `SetUp`, kar pomeni da se bo izvedla pred vsako testno metodo (slika 4.12).

**xUnit**

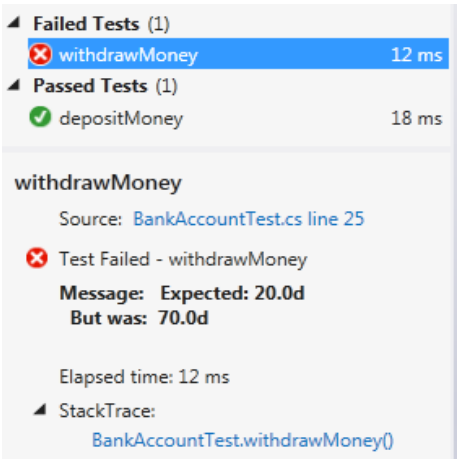
xUnit je za razliko od ogrodja NUnit drugače zasnovan, saj se vsaka testna metoda izvede v novi instanci testnega razreda. Ogrodje nam tako omogoča večjo izoliranost testnih metod, kar je tudi ena izmed glavnih zahtev sodobnega ogrodja za testiranje enot. Zaradi spremenjenega življenskega cikla v ogrodju xUnit, nam

```
namespace NUnit
{
    [TestFixture]
    class BankAccountTest
    {
        private BankAccount account = new BankAccount();

        [Test]
        public void depositMoney()
        {
            account.deposit(50);
            double expected = 50;
            Assert.AreEqual(expected, account.total);
        }

        [Test]
        public void withdrawMoney()
        {
            account.deposit(50);
            account.withdraw(30);
            double expected = 20;
            Assert.AreEqual(expected, account.total);
        }
    }
}
```

Slika 4.10: Testni razred v ogrodju NUnit, ki za vse testne metode uporablja isto instanco testnega razreda




Slika 4.11: Rezultati testiranja zgornjih testnih metod

```
namespace NUnit
{
    [TestFixture]
    class BankAccountTest
    {
        private BankAccount account;

        [SetUp]
        public void Init()
        {
           account = new BankAccount();
        }
        //sledijo testne metode
```

Slika 4.12: Pravilno napisan testni razred z uporabo atributa “SetUp“

testna metoda, ki se v ogrodju NUnit ni uspešno izvedla, tukaj uspešno izvede brez uporabe atributov. Ob koncu testne metode se trenutna instanca testnega razreda enostavno uniči.

**MSTest**
MSTest je tako kot ogrodje xUnit zasnovan z zahtevo po neodvisnosti metod, saj se vsaka testna metoda izvede v novi instanci testnega razreda, kar pomeni da uporaba posebnih atributov kot pri ogrodju NUnit ni potrebna.

**NUnit**:

- vse testne metode tečejo v isti instanci tesnega razreda
- za zagotavljanje neodvisnosti je obvezna uporaba atributov
- potrebna dodatna previdnost pri zagotavljanju neodvisnosti metod

**xUnit**:

+ za vsako testno metodo se ustvari nova instanca testnega razreda
+ vsaka testna metoda je neodvisna od prejšnjih metod

**MSTest**:

+ za vsako testno metodo se ustvari nova instanca testnega razreda
+ vsaka testna metoda je neodvisna od prejšnjih metod

| NUnit | xUnit | MSTest |
|---|---|---|
| 3 | 5 | 5 |

Tabela 4.8: Ocene ogrodij za kriterij "način izvajanja testiranja"

### 4.1.8 Razvoj ogrodja

Eden izmed pomembnejših kriterijev pri izbiri ogrodja je seveda tudi njegov razvoj. Z vse hitrejšim in zahtevnejšim razvojem programske opreme, se morajo tudi ogrodja za testiranje temu prilagajati. Ogrodje, ki je bilo najboljše pred nekaj leti, je lahko danes že zastarelo in neprimerno za učinkovito testiranje. Pri kriteriju bomo preverili ali so izbrana ogrodja še v fazi aktivnega razvoja, ali razvijalci uvajajo nove funkcije in sledijo smernicam testiranja programske opreme.

**NUnit**

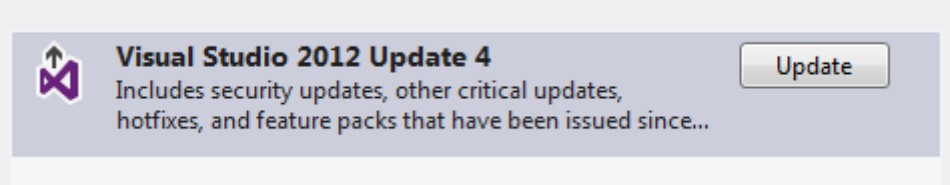
Ogrodje NUnit je bilo leta 2002 eno izmed prvih resnih orodij za testiranje enot v .NET programskih jezikih. Razvoj je bil aktiven, vendar je ogrodje v zadnjem času v fazi stagnacije. Trenutna stabilna verzija ogrodja (verzija 2.6.4) je bila izdana 16. decembra 2014, verzija 2.6.3 pa oktobra 2013. Večjih sprememb v novih verzijah ni, večinoma najdemo samo popravke.
Od decembra 2014 je na uradni spletni strani na voljo verzija 3.0, v kateri so razvijalci najavili korenite spremembe, ki pa žal še ni stabilna.

**xUnit**

Zadnja stabilna verzija ogrodja xUnit (verzija 1.9.2) je bila objavljena 26. avgusta 2013. Od sredine leta 2012 pa lahko najdemo tudi verzijo 2.0, ki pa še ni primerna za vsakodnevno uporabo. V času pisanja diplome je zadnja posodobitev za verzijo 2.0 iz maja 2014.

**MSTest**

Sledenje novim posodobitvam je najtežje pri ogrodju MSTest, saj uradna spletna stran posebej za orodje ne obstaja. Novosti se tako namestijo preko popravkov, ki jih Microsoft izda za Visual Studio (slika 4.13). Trenutno najnovejša posodobitev je iz začetka novembra 2013 (Visual Studio 2012 Update 4).




Slika 4.13: Pojavno okno za posodobitev razvojnega okolja Visual Studio

**NUnit**:

+ v pripravi naj bi bila verzija 3.0

**xUnit**:

+ aktiven razvoj
+ v pripravi je verzija 2.0

**MSTest**:

- slab pregled nad posodobitvami
- večjih posodobitev za ogrodje od same izdaje ni bilo

| NUnit | xUnit | MSTest |
|---|---|---|
| 4 | 4 | 3 |

Tabela 4.9: Ocene ogrodij za kriterij "razvoj ogrodja"

### 4.1.9 Uporabnost v praksi

Pri zadnjem kriteriju smo si želeli preveriti kako se ogrodja odrežejo na praktičnem primeru. Napisali smo kratek program (slika 4.14), katerega smo stestirali z vsemi tremi ogrodji in tako pridobili dodaten vpogled v delovanje ogrodij v praksi. Poleg tega so nas zanimala tudi mnenja ostalih uporabnikov, zato smo pobrskali po forumih in zbrali nekaj vtisov drugih razvijalcev.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LinkedList {

    class Seznam {
        Node head;
        Node current;
        private int size;

        public Seznam() {
            head = null;
            size = 0;
        }

        public int count() {
            return size;
        }

        public void addLast(string value) {
            if (head == null) {
                Node n = new Node();
                n.value = value;
                head = n;
                current = head;
            }
            else {
                Node n = new Node();
                n.value = value;
                current.next = n;
                current = n;
            }
            size++;
        }

        public void addFirst(string value) {
            if (head == null) {
                Node n = new Node();
                n.value = value;
                head = n;
                current = head;
            }
            else {
                Node n = new Node();
                n.value = value;
                head = n;
                head.next = current;
                current = head;
            }
            size++;
        }
```

```
public void addPosition(string value, int position) {
    Node currentPosition = head;
    Node currentPositionLast = head;
    if (position <= 0)
        throw new NotImplementedException();
    else {
        for (int i = 1; i < position; i++) {
            currentPositionLast = currentPosition;
            currentPosition = currentPosition.next;
        }
        Node node = new Node();
        node.value = value;
        node.next = currentPosition;
        currentPositionLast.next = node;
        size++;
    }
}

public Node retrieve(int position) {
    Node temporaryNode = head;
    Node returnNode = null;
    int count = 1;

    while (temporaryNode != null) {
        if (count == position) {
            returnNode = temporaryNode;
            break;
        }
        count++;
        temporaryNode = temporaryNode.next;
    }
    return returnNode;
}
```

```
        public bool delete (int position) {
            if (position > 0 && position <= size) {
                if (position == 1) {
                    if (size == 1) {
                        head = null;
                        current = null;
                    }
                    else {
                        head = head.next;
                    }
                    size--;
                    return true;
                }
                else {
                    int cnt = 1;
                    Node temporaryNode = head;
                    Node lastNode = null;

                    while (temporaryNode != null) {
                        if (cnt == position) {
                            if (position == size) {
                                current = lastNode;
                            }

                            lastNode.next = temporaryNode.next;
                            size--;
                            return true;
                        }

                        cnt++;
                        lastNode = temporaryNode;
                        temporaryNode = temporaryNode.next;
                    }
                }
                return false;
            }
            else {
                throw new IndexOutOfRangeException();
            }
        }

        public void print() {
            Node temporaryNode = head;
            while (temporaryNode != null) {
                Console.WriteLine(temporaryNode.value);
                temporaryNode = temporaryNode.next;
            }
        }
    }
}
```

Slika 4.14: Testni program, na katerem smo stestirali vsa tri ogrodja

**NUnit**

Ogrodje NUnit se je pri testiranju izkazalo kot zelo konkurenčno in učinkovito orodje. Za delo z razvojnim okoljem Visual Studio je seveda potrebna namestitev, ki pa je hitra in enostavna. Grafični vmesnik je malenkost zastarel, ampak služi svojemu namenu. Večja pomankljivost se je pokazala pri zagotavljanju izoliranosti testnih metod, za kar smo bili primorani uporabiti atribut `[SetUp]`.

**xUnit**

Tako kot pri ogrodju NUnit, je bila tudi tukaj najprej potrebna namestitev ogrodja. Za zagotavljanje neodvisnosti metod uporabniku ni potrebno skrbeti, saj za to poskrbi že samo ogrodje. Izredno uporabna je vgrajena možnost izdelave poročil, katero pa smo lahko zagnali samo preko ukazne vrstice in ne v grafičnem vmesniku. Testne metode se lahko nahajajo v vsakem javnem razredu, potrebno jih je le označiti s pravilnim atributom. Napišemo jih lahko celo v istem razredu kot sam program, čeprav zaradi preglednosti to ni priporočljivo.

**MSTest**

Ogrodje MSTest je privzeto nameščeno v razvojno okolje Visual Studio, kar nam odstrani potrebo po dodatni namestitvi. Pogrešali smo edino možnost izdelave poročil brez nameščanja dodatnih orodij, tako kot nam to omogoča xUnit.

Po krajšem prebiranju prispevkov na forumih smo izmed primerjanih orodij kot največkrat priporočeno ogrodje zasledili NUnit. Priljubljenost je ogrodje pridobilo že ob samem izidu in je trenutno eno izmed najuporabnejših med .NET razvijalci. Zaradi njegove priljubljenosti lahko na spletu najdemo veliko število razširitev, ki jih lahko vključimo tudi v ostala razvojna okolja. Kot konkurenco ogrodju NUnit zasledimo xUnit in MSTest. xUnit je po priporočilih zelo blizu ogrodju NUnit. Mnenja za MSTest so pa različna. Nekateri menijo, da je solidno orodje katerega največja prednost je njegova integracija z okoljem Visual Studio, drugi pa uporabo odsvetujejo in priporočajo namestitev konkurenčnih ogrodij, med njimi tudi NUnit in xUnit.

**NUnit**:

+ največkrat priporočeno ogrodje
+ veliko število različnih razširitev tudi za ostala razvojna okolja
+ najbolj priljubljeno med .NET razvijalci

**xUnit**:

+ vgrajena izdelava poročil
+ po mnenjih uporabnikov tesno za ogrodjem NUnit

**MSTest**:

+ integriranost z razvojnim okoljem Visual Studio
- slabši odziv pri uporabnikih

| NUnit | xUnit | MSTest |
|---|---|---|
| 5 | 4 | 3 |

Tabela 4.10: Ocene ogrodij za kriterij "uporabnost v praksi"

## 4.2 Utežitev kriterijev

Uteževanje kriterijev je ključnega pomena za čim bolj objektivne rezultate, saj moramo imeti potrebno znanje za presojo o njihovi pomembnosti. Vsakemu kriteriju bomo dodelili neko utež, večja kot je utež, bolj pomemben je kriterij. Skupna vsota kriterijev bo znašala 100 točk.
Tako smo največjo utež namenili kriteriju "način izvajanja testiranja", saj je izredno pomembno razumevanje življenjskega cikla testnega razreda, ker lahko vpliva na rezultate testnih metod. Zaradi izdelave dokumentacije in poročanja o napredku je pomemben tudi kriterij "izdelava poročil", med pomembnejše pa smo uvrstili še "možnosti konfiguracije testov" in seveda jedro ogrodij - "trditvene metode". Ker pa se tehnologija hitro spreminja, smo večjo utež namenili tudi samemu "razvoju ogrodja".

**Obstoj dokumentacije** 10
**Popularnost** 7,5
**Integracija z razvojnim okoljem Visual Studio** 10
**Izdelava poročil** 12,5
**Možnosti konfiguracije testov** 12,5
**Trditvene metode** 12,5
**Način izvajanja testiranja** 15
**Razvoj ogrodja** 12,5
**Uporabnost v praksi** 7,5

## 4.3 Analiza rezultatov

Končne rezultate (tabele 4.11, 4.12, 4.13) smo dobili tako, da smo za vsako ogrodje izračunali vsoto zmnožkov ocene in uteži.

**NUnit**:

| **Kriterij** | **Utež** | **Ocena** | **Rezultat** |
|---|---|---|---|
| Obstoj dokumentacije | 10 | 5 | 50 |
| Popularnost | 7,5 | 5 | 37,5 |
| Integracija z razvojnim okoljem Visual Studio | 10 | 4 | 40 |
| Izdelava poročil | 12,5 | 3 | 37,5 |
| Možnosti konfiguracije testov | 12,5 | 4 | 50 |
| Trditvene metode | 12,5 | 4 | 50 |
| Način izvajanja testiranja | 15 | 3 | 45 |
| Razvoj ogrodja | 12,5 | 4 | 50 |
| Uporabnost v praksi | 7,5 | 5 | 37,5 |
| | | | **397,5** |

Tabela 4.11: Končni rezultat za ogrodje NUnit

**xUnit**:

| **Kriterij** | **Utež** | **Ocena** | **Rezultat** |
|---|---|---|---|
| Obstoj dokumentacije | 10 | 4 | 40 |
| Popularnost | 7,5 | 3 | 22,5 |
| Integracija z razvojnim okoljem Visual Studio | 10 | 4 | 40 |
| Izdelava poročil | 12,5 | 5 | 62,5 |
| Možnosti konfiguracije testov | 12,5 | 5 | 62,5 |
| Trditvene metode | 12,5 | 4 | 50 |
| Način izvajanja testiranja | 15 | 5 | 75 |
| Razvoj ogrodja | 12,5 | 4 | 50 |
| Uporabnost v praksi | 7,5 | 4 | 30 |
| | | | **432,5** |

Tabela 4.12: Končni rezultat za ogrodje xUnit

**MSTest**:

| **Kriterij** | **Utež** | **Ocena** | **Rezultat** |
|---|---|---|---|
| Obstoj dokumentacije | 10 | 2 | 20 |
| Popularnost | 7,5 | 4 | 30 |
| Integracija z razvojnim okoljem Visual Studio | 10 | 5 | 50 |
| Izdelava poročil | 12,5 | 3 | 37,5 |
| Možnosti konfiguracije testov | 12,5 | 4 | 50 |
| Trditvene metode | 12,5 | 4 | 50 |
| Način izvajanja testiranja | 15 | 5 | 75 |
| Razvoj ogrodja | 12,5 | 3 | 37,5 |
| Uporabnost v praksi | 7,5 | 3 | 22,5 |
| | | | **372,5** |

Tabela 4.13: Končni rezultat za ogrodje MSTest

Glede na rezultate je izmed testiranih ogrodij xUnit najboljše orodje za testiranje enot. Na drugem mestu zasledimo ogrodje NUnit, na zadnjem pa sledi ogrodje MSTest.

Zaradi daljše zgodovine je ogrodje NUnit prepričljivo zmagalo pri kriterijih za obstoj dokumentacije in pri popularnosti. Slabše se je odrezalo pri najpomembnejšemu kriteriju - načinu izvajanja testiranja, saj ogrodje brez uporabe atributov ne zagotavlja neodvisnosti med testnimi metodami.
xUnit je prepričljivi zmagovalec, saj nam omogoča izdelavo poročil brez dodatnih orodij, dokumentacija je pregledna in zagotavlja nam izoliranost testnih metod brez uporabe atributov.
Ogrodje MSTest izstopa po samo dveh kriterijih, pri zagotavljanju neodvisnosti metod in zaradi njegove integriranosti v razvojno okolje Visual Studio.

Zaradi prepričljive zmage ogrodja xUnit bi za prevlado drugega ogrodja bila potrebna dokaj velika sprememba uteži. Tako bi dosegli izenačenje med ogrodjema NUnit in xUnit, če bi za 5 točk povečali uteži kriterijema “obstoj dokumentacije“ in “popularnost“, skladno s tem znižali uteži kriterijema “izdelava poročil“ in “način izvajanja testiranja“. Za zmago ogrodja NUnit pa bi poleg zgornjih sprememb bilo potrebno še vsaj za 2,5 točke povečati utež kriteriju “uporabnost v praksi“.
Ogrodje MSTest bi se lahko s povečanjem uteži pri kriterijih kot so “integracija z razvojnim okoljem Visual Studio“ in “način izvajanja testiranja“ lahko zavihtelo na drugo mesto, torej pred ogrodje NUnit. Za zmago pa bi bilo treba uteži kriterijev kjer ogrodje prevladuje precej povečati.

# Poglavje 5

# Sklepne ugotovitve

Ob izdelavi diplomske naloge smo ob začetnem prebiranju literature ugotovili, da je testiranje programske zelo obširno področje. Tako smo bili najprej primorani pridobiti osnovna znanja o procesu testiranja, kjer smo tudi spoznali kako pomembna je ta aktivnost.
V diplomski nalogi smo se posvetili primerjavi treh ogrodij za testiranje enot. Rezultati so razkrili prednosti kot tudi slabosti vsakega ogrodja.

Absolutni zmagovalec je ogrodje xUnit, ki bi ga priporočili vsem razvijalcem, saj je izmed orodij, ki smo jih primerjali, zaradi odlične zasnove dosegel največje število točk.
NUnit je zaradi odlične dokumentacije, dolge zgodovine in njegove popularnosti pristal na drugem mestu in ga lahko smatramo kot odličnega konkurenta ogrodju xUnit. Pozorni pa moramo biti pri zagotavljanju izoliranosti metod.
Ogrodje MSTest je zaradi integracije z razvojnim okoljem Visual Studio odlična izbira za začetnike, ki želijo spoznati osnove testiranja enot. Nekaj težav se bo vseeno pojavilo pri prebiranju in iskanju dokumentacije.

Ogrodja smo poizkušali oceniti kar se da učinkovito in objektivno, neodvisno od naših želja. Za še boljšo primerjavo bi lahko testirali več različnih ogrodij in mogoče tudi dodali kakšen nov kriterij.

# Literatura


[1] P. Ammann, J. Offutt, "Introduction to Software Testing", New York: Cambridge University, 2008

[2] "Avtomatizacija testiranja", dostopno na:
http://en.wikipedia.org/wiki/Test_automation (19.4.2014)

[3] "Bing", dostopno na:
http://www.bing.com (3.5.2014)

[4] "Citati o programskem testiranju", dostopno na:
http://softwaretestingfundamentals.com/software-testing-quotes/
(11.1.2015)

[5] "Google", dostopno na:
https://www.google.si (3.5.2014)

[6] "Integrirano razvojno okolje", dostopno na:
http://en.wikipedia.org/wiki/Integrated_development_environment
(30.5.2014)

[7] "Kratka zgodovina programskega testiranja", dostopno na:
http://www.slideshare.net/UniversalExams/basic-history-of-software-testing
(19.4.2014)

[8] "NUnit", dostopno na:
http://www.nunit.org/ (3.5.2014)

[9] "Ogrodje", dostopno na:
http://www.techterms.com/definition/framework (3.5.2014)

[10] "Razhroščevanje", dostopno na:
http://searchsoftwarequality.techtarget.com/definition/debugging
(21.3.2014)

[11] "Testiranje enot ali integracijsko testiranje", dostopno na:
http://ardalis.com/unit-test-or-integration-test-and-why-you-should-care
(28.4.2014)

[12] "Testiranje z metodo bele škatle", dostopno na:
http://en.wikipedia.org/wiki/White-box_testing (19.4.2014)

[13] "Testiranje z metodo črne škatle", dostopno na:
http://en.wikipedia.org/wiki/Black-box_testing (19.4.2014)

[14] "Trditveni razredi", dostopno na:
http://msdn.microsoft.com/en-us/library/ms182530(v=vs.80).aspx
(24.5.2014)

[15] "Verifikacija in validacija", dostopno na:
http://en.wikipedia.org/wiki/Verification_and_validation (21.3.2014)

[16] "xUnit", dostopno na:
https://github.com/xunit/xunit (8.5.2014)

[17] "Yahoo", dostopno na:
https://www.yahoo.com (3.5.2014)